Preliminary Definition of CORTEX Interaction Model

G. Biegel, C. Brudna, A. Casimiro, J. Kaiser, C. Liu,
C. Mitidieri and P. Veríssimo

DI-FCUL TR–03–16

July 2003

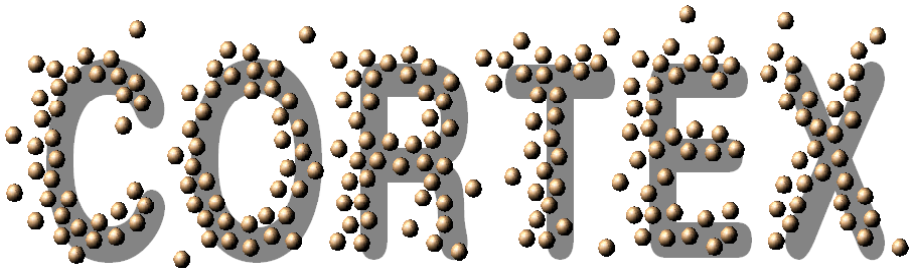
Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1700 Lisboa
Portugal

Technical reports are available at http://www.di.fc.ul.pt/tech-reports. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Project IST-2000-26031

CO-operating Real-time senTient objects:

architecture and EXperimental evaluation



Preliminary Definition of the Interaction Model

CORTEX deliverable D3

Version 1.0

March 27, 2002

Revisions

Rev.	Date	Comment
0.1	15/02/2002	Integration of partner contributions
1.0	27/02/2002	Final version

Editor

Jörg Kaiser, University of Ulm

Contributors

Greg Biegel, TCD
Cristiano Brudna, University of Ulm
Antonio Casimiro Costa, University of Lisbon
Jörg Kaiser, University of Ulm
Changling Liu, University of Ulm
Carlos Mitidieri, University of Ulm
Paulo Jorge Verissimo, University of Lisbon

Address

Department of Computer Structures,
University of Ulm,
Germany

Table of Contents

1 Introduction

The objectives of this work package are to define the interaction model for co-operating sentient objects [Cor01]. As it is detailed in [Cor02], there are a number of challenging applications, which need adequate interaction models supporting dynamic system evolution and proactive user independent activities over a heterogeneous network environment. The interaction model therefore is a central issue as well for the programming model level as it is for the middleware. On the programming level, an event-based interaction model has been adopted. The focus of this work package is to define the abstractions provided in the infrastructure to support the higher-level notion of events and to specify the functional and non-functional aspects of these abstractions. Interaction comprises the aspects of communication and co-ordination. Communication defines the basic way in which sentient objects exchange information in the event-based model. Co-ordination addresses issues of agreement and synchronization of joint activities. One of the novel views on interaction in CORTEX is the inclusion of the communication and co-ordination through the environment. Fig 1. depicts this interaction between the technical artefacts and the environment [Cor01].

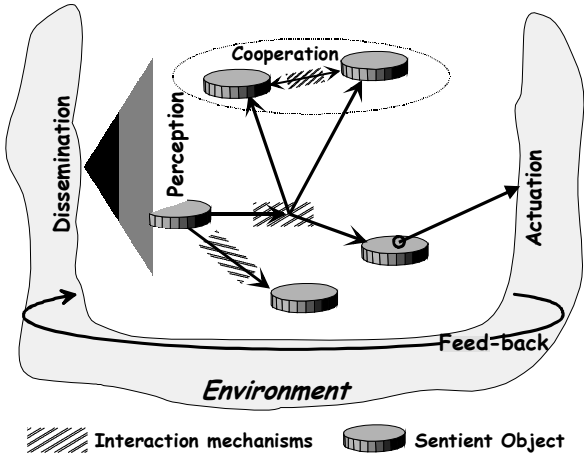


Fig. 1 Events and object interactions in CORTEX

Therefore, the definition of the interaction model has to be general enough to cover a wide range of possible communication channels. These channels may be realised in a networked technical environment or also as channels between certain actuators and specialized sensors that are able to capture this actuation on the environment. The physical environment then would take the role of a channel mediating the information through a physical carrier like e.g. light, sound, or a chemical substance.

Chapter 3 introduces the basic requirements for interacting sentient objects as defined in WP1-D2 and provides a preliminary definition of event channels. The focus here is on a technical network infrastructure, however, the basic definitions are broad enough include other communication substrates. The requirements are on the one side derived from the definition of the programming model in WP1-D2. On the other side, the requirements come from the envisaged highly dynamic and co-operative nature of the system, which is composed from a large number of nodes, which may comprise sensors and actuators.

Chapter 4 explicitly describes a way of co-ordinating sentient objects through the environment. The technical term "stigmergy" was coined by a biologist who used it to describe the co-ordination of populations of insects without direct communication between the individuals. This mapped to a model suited for the CORTEX context.

Any activity which is carried out in the physical world needs to adapt to the pace and dependability requirements dictated by the environment. In technical terms this means that non-functional properties of the system, as timeliness and reliability of operation has to be included. These Quality of Service (QoS) attributes have to be guaranteed even in an environment where unanticipated dynamic change is one of the inherent properties. Chapter 5 introduces an adaptive QoS mechanism based on a reliable and timely system service. This service, called the Timely Computing Base (TCB) is able to monitor distributed system activities and to provide an "early warning system" for temporal and functional failures. Additionally, important distributed system functions can be executed by this small trusted kernel.

The TCB is exploited in combination with time- and failure elastic applications to achieve a predictable behaviour even in the presence of unexpected network delays and late or missing information. It is in line with the autonomy postulates in CORTEX which gives high priority to mechanisms which allow taking decisions locally based on the awareness what is going on in the system. Chapter 6 explains how this can be achieved.

[Cor01] CORTEX – “Annex 1, Description of Work”, October 2000.

[Cor02] CORTEX – “Definition of Application Scenarios”, October 2001.

2 Communication Abstractions in CORTEX

Jörg Kaiser
Carlos Mitidieri

University of Ulm

This chapter introduces the basic requirements for interacting sentient objects as defined in WP1-D2 and provides a preliminary definition of event channels. The focus here is on a technical network infrastructure, however, the basic definitions are broad enough include other communication substrates. The requirements are on the one side derived from the definition of the programming model in WP1-D2. On the other side, the requirements come from the envisaged highly dynamic and co-operative nature of the system which is composed from a large number of nodes which may comprise sensors and actuators.

Communication Abstractions in CORTEX

Jörg Kaiser
Carlos Mitidieri

University of Ulm

2.1 Requirements on the interaction model

2.1.1 Basic System Assumptions

One of the basic assumptions made in CORTEX is that technological advances encourage the design of systems which are composed of a large number of smart components which in the one or the other way are sensing their environment or acting upon it. These components may comprise mechanical components, hardware and software and can be of low or high complexity depending on the specific application task (an example of such a component, the optical sensor ICU is presented in WP1-D2). Their main property from a system's point of view, is the inclusion of an active processing element and the encapsulation of a certain functionality which is accessible via a network interface. This interface shields the system from the low level functional and temporal details of controlling a specific sensor or actuator. Fig. 1 gives an example of such a system, an autonomous robot that is equipped with smart sensors and actuators. The smart components communicate with other components via a hierarchy of networks as it is further detailed in WP3-D4. Wireless connections enable mobility. Subsequently, this example will be used to explain the requirements for the interaction.

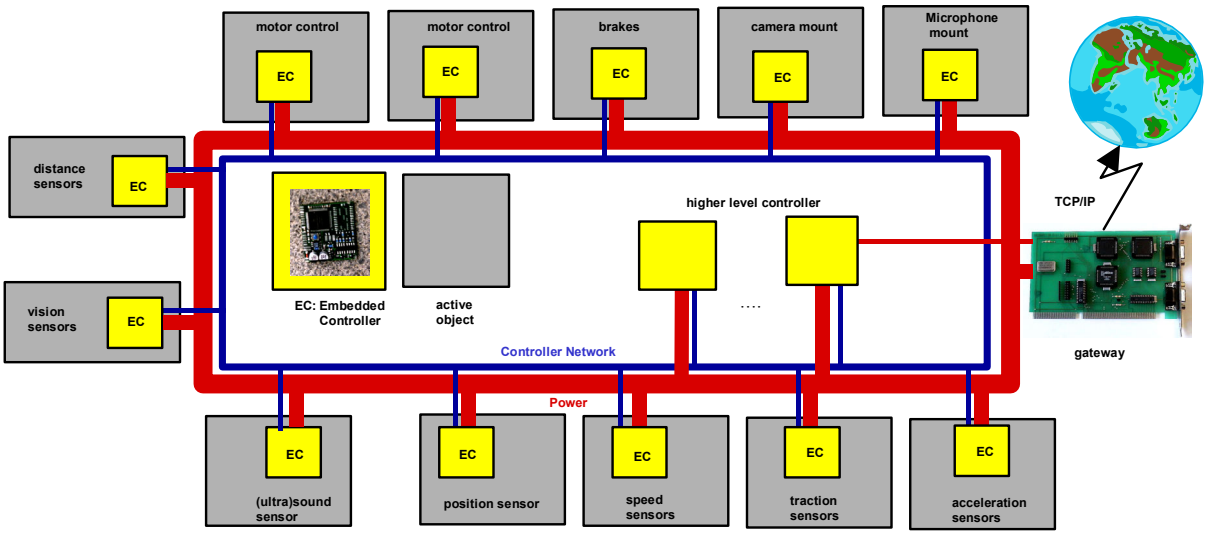


Fig. 1 Example of a sensor/actuator network

The built-in computational components enable the implementation of well-defined high level interfaces that do not just provide raw transducer data, but pre-processed, application-related information**.** Consequently, the interfaces and the functions of these smart components may include functions related to overall control, supervision, and maintenance issues. The most interesting and challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture in which smart autonomous components co-operate to control physical processes co-operatively without the need of a central co-ordination facility. In such a system, multiple different sensors will co-operate to augment perception of the environment and autonomous actuators will co-ordinate actions to increase speed, power and quality of actuation thus forming decentralized virtual sensor and actuator networks. On a lower system level, these components can be viewed as smart transducers [KHE00], [OMG00]. However, to exploit the full advantages of this system model, a higher-level view is necessary. Here the notion of active sentient objects which are the basic entities of the programming model matches well the requirement of a high-level system model. This model is introduced in WP1-D2. Subsequently, we use the term sentient object to refer to these system components.

2.1.2 Basic Assumptions and Requirements for the Interaction Model

Interaction comprises a communication and a co-ordination aspect. While communication refers to information dissemination, co-ordination specifically addresses control issues. Because in CORTEX the autonomy of sentient objects is seen as a major design objective, it is important to treat these aspects orthogonal. Based on these considerations we can define the following assumptions and requirements for the interaction between sentient objects. :

1. **Sentient objects are autonomous**. Autonomy means that each sentient object is in its own sphere of control and no control transfer should cross the boundary of the component. As a consequence, sentient objects can only interact on the basis of shared information.
2. **Communication is asynchronous.** This is a direct consequence of 1. The need for control autonomy requires that an information producer should never block because of a control transfer. Hence communication must be asynchronous[1]. Asynchronous communication is the prerequisite to keep coordination and communication independently.
3. **Interaction is spontaneous.** Sentient objects have to react to external events. These external events are recognized at the sensor interface of an sentient object at arbitrary points in time and lead to spontaneous communication activities to disseminate the information. This is best captured in a generative, event-based interaction pattern rather than in a client server model.

[1] The terms **synchronous** and **asynchronous** are somehow overloaded and used in many ways, e.g. to refer to temporal properties of the communication system. Here it only refers to the non-blocking property of communication.

4. **The architecture should support many-to-many communication patterns.** A typical situation is that the information provided by a sentient object will be used and analysed in more than one place. E.g. the information disseminated by the optical sensor ICU (cf. WP1-D2) on a mobile robot is interesting for reactive motor control as well as for long-term navigation. Another typical example is the situation in which control information issued from a sentient object addresses a number of identical actuators; e.g., all motors have to stop in case of an emergency.
5. **Communication is anonymous.** Consider again the example of stopping a set of motors. When issuing a stop command, it is not of interest to address a specific motor, rather it must be ensured that all relevant motors receive the command. Similarly, when reacting to a stop command, it is not of interest which controller has issued that command. On a more abstract level, a sentient object triggered by the progression of time or the occurrence of an event spontaneously generates the respective events and distributes it to the system. Thus, it is considered as a producer of information. The corresponding consumer objects have filters to determine whether this information is useful for them. This interaction leads to a model of anonymous communication in which the producer does not know which consumers will use its information and, vice versa, the consumers only know which information they need independently from which source they receive it.

   From these properties of anonymous communication, an incremental and component oriented system design can be derived. At design time, when a component is defined, no communication relations have to be made explicit, i.e. it is not required to define the communication participants. Rather it has to be defined what will be communicated, i.e. the content of a message has to be represented in some form. Therefore, anonymous communication supports the extensibility and the reliability of the system because objects can be added or be replaced easily without changing address information maintained in the other objects.

2.1.3 Interaction models

The interaction model supported by the system middleware should meet the requirements enumerated above. We will briefly revisit the common models of interaction found in the communication abstractions of popular middleware. We will provide a short overview and a comparative analysis of these communication models in terms of both, the above stated requirements and the impact on overall system architecture. A detailed discussion and a comparative case study can be found in chapter 3 ([PBV01]).

Remote Method Invocation (RMI). This is the prevailing model used in distributed object-oriented middleware. It is directly derived from the local method call. Its main goal is to preserve the same programming model therefore in a distributed system it provides a completely transparent way of remote object interaction. There are a couple of properties directly related to this model:

RMI supports a client-server interaction model.
In this model, a client requests information from a server. The model supports:

- A client-server relation for a particular invocation
- One-to-one communication
- Synchronous communication
- Early binding of a client to a server

The client-server relation and the one-to-one communication are in conflict with the requirements stated above. Secondly, synchronous communication, although efficient in a local system, has severe drawbacks in the distributed case and suffers from long, unpredictable blocking times. The usual semantics associated with an RMI is that after the object performed the invocation, its execution flow is blocked and the control is transferred to the invoked server method. This may be enforced by a remote invocation that leads to invocation chains invisible to the client which initiated the request. It therefore also violates the requirements derived for autonomous sentient objects. Thirdly, before an object can communicate and issue an RMI it must have a reference to the respective server object. Hence at design time, an early explicit relation among interacting objects must be defined. This contradicts the anonymity requirement useful for an incremental, component-centred system evolution. Therefore, RMI seems to be less suited to support the interaction between sentient objects.

Port based communication. A port is an abstraction which stems from the operating system area. Recently, it has received attention in the context of component-oriented software engineering for connecting object instances. A protocol is associated to each port, which describes the set of valid messages and their respective directions (i.e. incoming or outgoing). Hence, one component can be designed without any prior knowledge about which objects are going to interact. The binding (or conjugation) between any two ports (two ports are called conjugated when outgoing messages from one are accepted as incoming messages in the other) can be deferred to the system configuration time. Ports also support a basic one-to-many communication model, because many incoming ports may be connected to one outgoing port. At runtime this configuration is static. When a component sends message, it has to write to the related port without any knowledge who will receive the message. Vice-versa, when the component needs specific information, it reads the data from the indicated port not caring about who sent it.

The port-based communication meets many of the requirements expressed above but it lacks some flexibility during run-time. Normally, the information which is necessary to perform the binding of outgoing and incoming ports is only available during configuration time. At this time, these relations are mapped to a set of addresses which are statically used by the underlying communication system during run-time. Hence it is not possible to change the configuration dynamically, e g when new mobile entities enter the system.

Interaction via a shared information space. It is worth noting, that co-ordination can either be accomplished by explicit control flow or by exploiting common knowledge, i.e. a common view on the environment and of what has to be achieved. In a dynamic system where autonomy of objects is one of the main objectives, and objects may dynamically join and leave a joint activity, explicit control flow is not an appropriate model for co-ordination. Therefore models of a shared information space in which decisions are autonomously made by the co-operating objects based on common knowledge have been developed. Examples are the data field architecture of autonomous decentralized systems (ADS) [Mor93], developed for large control systems, various blackboard architectures used for co-ordinating intelligent objects in robotics and AI, the tuple space in Linda [CaG89] and the Java Spaces model. The basic property of a shared information space is a complete decoupling of the producers of information and the consumers. A producer generates an event and puts the respective information in this space and the consumers can take whatever information is needed. No control flow is imposed to this many-to-many communication facility. A content-based scheme is used to maintain the dynamic communication relations. While this maps well to the requirements concerning generative, anonymous communication and the event-based programming model, co-ordination needs additional support to relate the occurrence of an event, which results in an update of information in the shared space to the point in time when it has been updated and secondly inform the interested consumers from this event occurrence. This notification service requires some protocol which maintains the relations between the producers of events and the consumers. Such protocols usually are termed publisher/subscriber protocols.

The publisher/subscriber model. This model extends the model of a shared information space by a notification facility. As a producer, an object may spontaneously publish an event. Objects interested in this event may subscribe. Notification of subscribers is an inherent function of the protocol, i.e. all objects that have subscribed to a certain event are notified when it is published. As the publisher/subscriber protocol is based on a shared information space model, it uses a content-based scheme to identify events. In the basic model, there is a single communication channel which is used by all publishers to disseminate their information. A direct realization of this model requires that the subscribers get notified whenever an update occurs and have to filter the relevant events locally. An alternative to this is the introduction of **event channels** which are related to a certain type of event. This structures the entire event space in subsections to which an object may subscribe. In this way, some pre-filtering of events is performed on the side of the publishers. As pointed out in [KaM99], this leads to a slight reduction of flexibility because content has to be bound to an event channel before it can be communicated but there are substantial benefits when implementing the scheme. These issues which particularly address the routing, filtering and binding trade-offs with respect to the underlying network are presented in the subsequent section.

The publisher/subscriber protocol seems to be particularly suited for the interaction patterns assumed in CORTEX. On the one side, it addresses the high interoperability demands by a model of a shared information space on the other hand, it includes a

notification property which seems to be necessary in an active event-based system model.
Table 1 summarizes the properties of the discussed interaction models. Column 3 and 4 contain the specific properties of a publisher/subscriber protocol based on a single shared channel and one exploiting multiple event channels.

	RMI	*Port Based*	*Single Channel*	*Multiple Event Channels*
Model	Client/ Server	producer/ consumer	producer/ consumer	producer/ consumer
Routing	Object name or address	Port name or address	Event content	Event type
Binding	Design time	Configuration time	No binding	Run time
Control transfer	Yes	No	No	No
Comm. Relation	Point-to-point	Point-to-point	Broadcast	Multicast and Broadcast
Filtering	Sender side	Sender side	Receiver side	Sender/Receiver side

Table 1. Summary of communication characteristics.

2.2 Event Channels

An event channel is a high level abstraction of a one-way communication channel connecting one or more producers of events to the consumers (Fig.2). It is characterised by the type of events it carries. It is a means to structure the global event space in content related sub areas. An event channel is identified by a subject which is a mapping from some identifier to a subset of events in the global space.

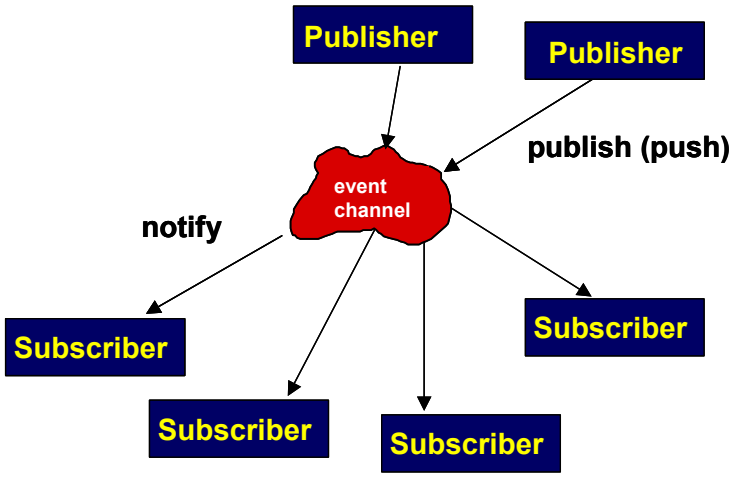


Fig.2 The event channel connecting publishers and subscribers

This subset corresponds to a certain type or class of information. The subject identifier must be unambiguous to allow the consumers to identify the respective event channel. The subject identifier can be represented e.g. by a textual description of the event or some unique number.

The notion of an event channel is rather general to describe the interactions in a technical network as well as the interactions through the environment as a real-world channel described in chapter 3. In the latter case, the channel will be defined by an actuator which pushes an event in the form of mechanical, electrical or chemical process energy to the channel. The channel mediates the energy to the subscribers which by means of specific sensors filter and receive the events.

Different from, e.g. object invocations which are closely related to a sender and a receiver and are inherently volatile, the representation of an anonymous event needs some substrate of its own. An event may live in the information space independently from producers or consumers and may have QoS attributes like age or quality associated with it. An event can be used without knowing the generating objects, even the lifetime of events can be seen independent from the generating entity. The event channel as a middleware abstraction provides the possibility to support this independence from producers and consumers. The event channel may have properties and attributes related to the non-functional requirements of event dissemination and can provide certain specified delivery semantics. E.g. in a control system, the relevance of an event is dependent on its age.

2.2.1 Exploring the Design Space for a Publisher/Subscriber Protocol

In this chapter we will concentrate on the technical issues on how the notion of event channels can be realized in a distributed way. We will discuss the main design trade-offs particularly with respect to temporal aspects and the fact that we may have networks with limited bandwidth and components with low processing capability.

On the level of the publisher/subscriber model we have objects as publishers and subscribers of an event, the information instance of an event, and the information type represented by an event channel. These abstractions of the model have to be mapped to the underlying to the elements provided by the technical infrastructure of the system such as objects, messages and addresses. More precisely, we can identify publishers and subscribers with (sentient) objects, and information instances with messages that are sent to certain addresses. The information type can be mapped to the address or the content of the message (or a combination of both). We have to solve the following problems:

- How can event channels be identified by publishers and subscribers?
- How are messages routed and filtered in a content-based fashion?
- How can content be bound to an addressing scheme?

As pointed out above, the most general approach to a publisher/subscriber protocol is the provision of a single channel and content-based addressing [CaG89]. In this case,

no identification of a channel is necessary. However, this approach puts the highest demands on the ability of the subscribers to filter the entire message stream by examining the content of every message. In an environment where smart sensors and actuators equipped with embedded micro controllers have to participate in the communication this will be infeasible. An optimization of the content-based scheme is the introduction of subject-based filtering in [OPS93]. The subject-based addressing maps well to the event channel concept, because a subject may identify a certain type of events and hence, a one-to-one relation to an event channel. However, in their "Information Bus ", although filtering is easier because only the subject of the message has to be evaluated, still each receiver has to examine every message. To meet the requirements imposed by the anticipated sensor/actuator network, filtering of messages has to be performed with a minimal overhead in the nodes. We therefore propose the dynamic binding of the network independent information type (which reflects the content of a message) to the addressing mechanism of the underlying communication infrastructure.

One of the big advantages which are highlighted for content- and subject-based addressing is just the absence of any binding. It is possible to communicate without having to know an address. Thus, a component which dynamically joins a network can immediately start communication. This property supports dynamic scalability and extensibility of a system. Binding in a way contradicts to the requirement of easy on-line extensibility, anonymity and independent component design as described above. However, if it is feasible to bind the content to the addressing mechanism of the underlying network, it is possible to exploit the network hard/ and firmware for filtering. Only those messages arrive at a node, to which objects on this node have subscribed. This would free the node from the tedious task of explicitly examining every message. The binding mechanism therefore should be:

1. Transparent to the object which publishes events or subscribes to events.
2. Lazy and dynamic, i.e. binding should occur at the latest possible moment during run- time by the publisher/subscriber middleware.

Fig. 3 depicts the basic components needed for a publisher/subscriber protocol which supports dynamic binding.

The Event Channel Handler (ECH) is the local component to provide the abstraction of event channels to the application objects. When an application objects subscribes to an event channel, it presents some identification of the event type (this is detailed in the description in WP3-D4). The ECH now creates a local representation of an event channel and subsequently performs the dynamic binding. We assume a service in the network, the Event Channel Broker (ECB), which is responsible to assign network dependent addresses to event channels. A more detailed architectural definition of the service is presented in WP3-D4. It should be noted that the ECB is not necessarily a central component as schematically depicted in Fig. 3. but the service could be provided by some consensus protocol. The ECB returns a network address to the ECH which now is able to configure the local network controller respectively. Whenever a message with the respective address is put on the network, the controller will automatically detect this without any further of higher software layers. Thus, also small controllers which are equipped with a network interface can

transfer the burden of filtering to a great part to the network hardware. An architectural design of the publisher/subscriber protocol is provided in WP3-D4.

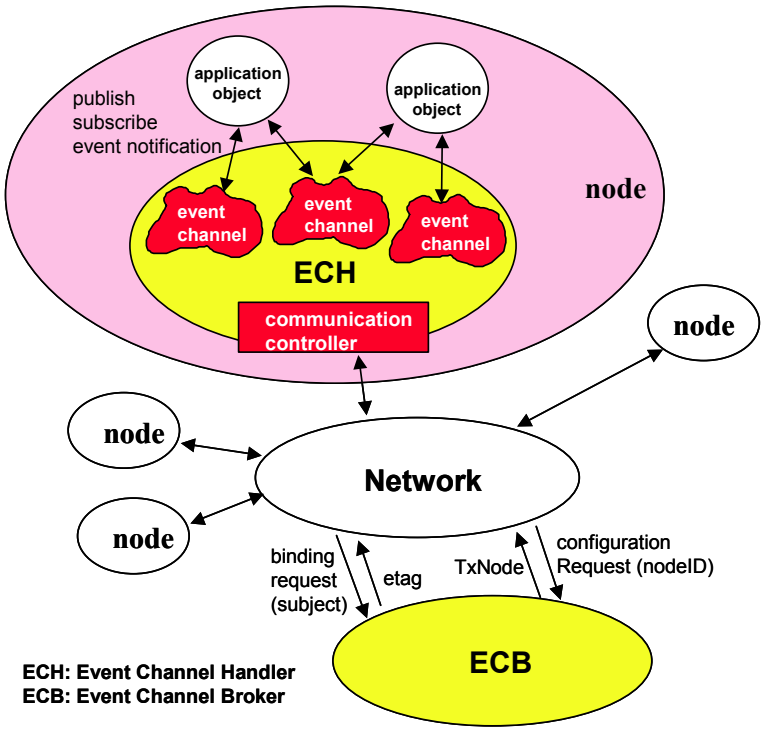


Fig. 3 Components of a publisher/subscriber middleware

2.3 References

[OMG00]
Object Management Group: "Smart Transducer Interface", Request for Proposal, OMG Document: orbos/2000-12-13, Dec. 2000

[KHE01]
H. Kopetz, M. Holzmann, W. Elmenreich:"A Universal Smart Transducer Interface: TTP/A", *Int. Journal of Computer System Science & Engineering, 16(2), March 2001*

[PBV01]
C.E.Pereira, L.B. Becker, C. Villela, C. Mitidieri, J. Kaiser
On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects
Proc. of the IEEE 4th International Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany, May 2001

[Mor93]
K. Mori. Autonomous decentralized Systems: Concepts, Data Field Architectures, and Future Trends, Int. Conference on Autonomous Decentralized Systems (ISADS93), 1993

[CaG89]
N. Carriero, D. Gelernter: "Linda in Context", Communications of the ACM, 32, 4, April 1989, pp 444-458.

[OPS93]
B. Oki, M. Pfluegl, A. Siegel, D. Skeen: "The information Bus®- An Architecture for Extensible Distributed Systems", 14th ACM Symposium on Operating System Principles, Asheville, NC, Dec 1993,pp.58-68.

[KaM99]
J. Kaiser, M. Mock: "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)", Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99), Saint-Malo, France, May 1999

3 On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects

C. E. Pereira , L. B. Becker, C. Villela, C. Mitidieri,
Federal University of Rio Grande do Sul (UFRGS) - Brazil[2]

J. Kaiser
University of Ulm - Germany[3]

The paper compares interaction and communication mechanisms used in distributed control systems, focusing on object-oriented and component-based development. The standard communication model used in distributed object-oriented systems is the remote method invocation. We argue that this client/server oriented model has some severe drawbacks when used in a control system where objects may have to broadcast information, spontaneously communicate environmental changes and where control autonomy is a crucial requirement. Therefore, we compare the traditional way of object invocation with a port-based scheme and the model of event channels. An application scenario from robot control is used to highlight similarities and differences among these mechanisms.

[2] This work has partly been supported by the German-Brazil Cooperation Scheme (WTZ mit Brasilien) under the contract: Projekt-Nr.: BRA 00/040 (ADOORATA)
[3] This work has partly been supported by the European Union's Information Society Technology Program under contract IST-2000.26031 (CORTEX)

On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects

C. E. Pereira , L. B. Becker, C. Villela, C. Mitidieri,
Federal University of Rio Grande do Sul (UFRGS) - Brazil[4]

J. Kaiser
University of Ulm - Germany[5]

Abstract: *The present paper compares interaction and communication mechanisms used in distributed control systems, focusing on object-oriented and component-based development. The standard communication model used in distributed object-oriented systems is the remote method invocation. We argue that this client/server oriented model has some severe drawbacks when used in a control system where objects may have to broadcast information, spontaneously communicate environmental changes and where control autonomy is a crucial requirement. Therefore, we compare the traditional way of object invocation with a port-based scheme and the model of event channels. An application scenario from robot control is used to highlight similarities and differences among these mechanisms.*

3.1 Introduction

Smart sensors and actuators, powered by micro-controllers and connected via a communication network support in many ways extensibility, reliability, and cost effectiveness of large control systems. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables. The communication network represents on a physical level a standardized interface over which the devices can exchange information. It can be foreseen that the technological advances will allow the integration of such smart devices as a single system-on-a-chip, which may comprise hardware, software and even mechanical components. In essence, each device becomes a configurable building block, which encapsulates data and behavior, can be configured by process specific parameters and communicates process relevant data. Consequently, the interfaces and the functions of these smart components are not just related to the raw physical values of the controlled device but they may include functions related to overall control, supervision, and maintenance issues. In such a system, multiple different sensors will co-operate to augment perception of the environment and actuators will co-ordinate actions to increase speed, power and quality of actuation thus forming virtual sensor and actuator networks. Perhaps the most challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture in which smart autonomous components co-operate to control a physical process without a central co-ordination facility. This property matches a vital requirement of many real-time systems for modularity and easy configuration to enable incremental system evolution. Particularly because the lifetime of real-time systems is much longer than the fast cycles of technology, it is of

[4] This work has partly been supported by the German-Brazil CooperationnScheme (WTZ mit Brasilien) under the contract: Projekt-Nr.: BRA 00/040 (ADOORATA)
[5] This work has partly been supported by the European Union's Information Society Technology Program under contract IST-2000.26031 (CORTEX)

utmost importance that components can be replaced without changing the entire system. Additionally, due to the continuous and uninterrupted operation requirement of large real-time systems, this reconfiguration should be made on-line.

The advantages of having such autonomous but cooperative components should be complemented by design methods able to exploit the envisaged distributed structure. Object-oriented design has proven to be one of the key technologies for the development of large and complex systems. The key concepts of modularization, information hiding, and inheritance gave decisive advantages over other software design methods. The emphasis is on the development process, incremental system extension, and better maintainability of software.

Among the existing methods for developing such distributed, usually embedded, computer-based systems, the adoption of the concept of distributed objects have been frequently mentioned in the literature as particularly adequate, specially due to their modularization and encapsulation characteristics, leading objects to be relatively self-contained and autonomous. This has an impact on how the interaction between entities has to be organized. The interaction between components comprises communication and co-ordination and is usually related to the overall system design method. In object-oriented systems, a prevailing way of interaction is through remote message invocation. Interaction thus is concerned with higher level issues and should provide an adequate abstraction from the underlying communication network. Since real-time aspects are a key issue when developing such systems, careful attention has to be paid to aspects related to their temporal behavior, which is strongly influenced by the communication patterns adopted.

Another attractive approach to promote flexibility and adaptability in distributed systems is the use of multi-agent systems. Like an object, an agent encapsulates state and methods, but different from it, an intelligent agent has the autonomy to decide at what time it attends to a request or even not to attend, although it could be forced by design to reply to pre-defined query types from particular agents. This introduces new dimensions to all aspects of system development, specially regarding communication. Agents work in a cycle of sensing-knowledge, processing-reasoning, and reaction-actuation. Messages from other agents should be accepted through internally controlled perceptual channels, in order to achieve a predictable timing. Furthermore, to enable the interaction among agents of different capabilities, communication must be defined at several levels, with less capable agents using more restrictive mechanisms. Others factors that influence interaction in multi-agent systems and may rise different needs are legacy software and system architecture. In the first case, a transducer agent could be implemented to translate agent queries into requests to existing programs. Finally, depending on the system architecture, agents can communicate directly or through an inter-mediator, with impact on the binding.

In this paper, three frequently adopted strategies for developing distributed real-time applications based on the concept of distributed objects are compared: remote method invocation, port-based communication, and publisher-subscriber. The paper aims to discuss the strengths and weaknesses of each approach both in terms of their adequacy for modeling as well as for implementing real-time systems based on distributed objects.

The remaining of this paper is organized as follows. The next section provides a more comprehensive description of each of the above mentioned communication strategies. A case study has been selected for comparing the different approaches. The case study

and the obtained results are presented on section three. Section four summarizes a comparison among the different strategies from the viewpoint of their adequacy to the development of distributed real-time systems. Section Five draws the conclusions and signals directions of future work.

3.2 Brief overview on selected communication mechanisms

When designing an interaction mechanism for a distributed object-oriented control system where distributed objects are considered as autonomous and concurrent processing units, some important features that should be supported by the underlying communication infra-structure are:

- Many-to-many communication - This is particularly important for control systems composed from intelligent sensors and actuators because the output of a sensor may be used by many entities.
- Spontaneous generation of messages triggered by a timer or by an external event.
- Control autonomy, i.e. co-ordination of activities is orthogonal to communication.
- Independent design and easy extensibility.
- Long term system evolution with incremental compilation possibilities.

In this section, the authors evaluate and compare different methods of high-level object interaction along this line. At first, the impact of many-to-many communication relations is examined. After that, it is discussed how spontaneous generation of messages is realized in the different methods. This is strongly related to the question of control autonomy. E.g. in a model relying on a request/reply scheme of interaction, communication and co-ordination are generally more strictly coupled as in a mechanism which supports an event driven model. Finally, design issues are discussed, with emphasis on at what time in the design process and how communication relationships (sender/receiver) have to be defined. This can be at design time, configuration time, run time or even can be omitted at all. Clearly, specifying communication relationships at design time makes it difficult to cope with a dynamically changing environment with respect to these relations because of the need to adapt to unanticipated events occurring during the mission of a system. Additionally, long term system evolution usually makes necessary the modification of old components or the creation of new ones. Therefore, it would be desirable to accommodate these changes "on-the-fly" on a running system without disturbing existing components.

3.2.1 Remote Method Invocation

In this communication pattern, as usually proposed by object-oriented languages such as C++ and Java, a client object must know (or using programming terminology must have a reference to) a server object. Remote methods calls resemble the remote procedure calls (RPC) in conventional distributed programming, with the difference that in object-oriented languages the called methods exist within the context of a given instance of a class.

This implies that an explicit bind –at design time - must be created between those elements that will interact. A sender object must know his receiver counterpart in order to be able to communicate. Internally, a remote method call is usually translated to a message, containing information related to the receiver object, the method to be invoked, and the method parameters. Message exchange then occurs using the underlying communication infrastructure. In order to have this mapping of method calls to messages as transparent as possible to applications, existing middleware based on distributed objects like CORBA and DCOM, introduces communication-specific objects, such as stubs, object adapters or proxies.

The remote method invocation thus characterizes a point-to-point (p2p) communication, in a client/server style, where the client object interface maintains a reference to the server object.

This interaction mechanism can execute either in a synchronous or in an asynchronous fashion. In the first case, a request-reply scheme is adopted, derived from the synchronous remote procedure call. Thereby, when one object invokes a method on another object, its execution flow is blocked and the control is transferred to the invoked method. On the other hand, when the interaction is asynchronous, an unblocking method call occurs, i.e. both sender and receiver objects may continue execution. Figure 1 depicts the exposed concepts.

One of the advantages of having the communication defined explicitly at design time is that a more rigorous consistency checking regarding the number and type of parameters transferred from sender to receiver can be performed. This can avoid some typical errors of distributed programming using only message passing mechanisms.

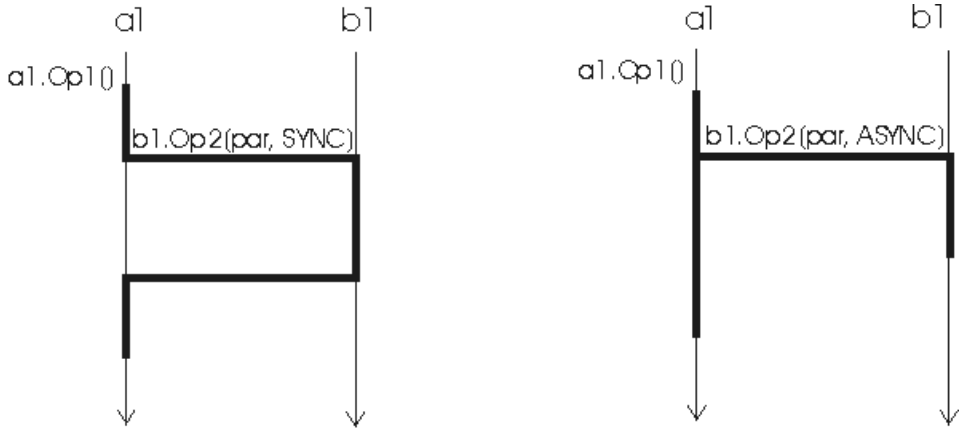


Figure 1: RMI interaction mechanism.

3.2.2 Port-based communication

Port-based communication: in this case, a component-oriented development is suggested, on which objects are interconnected and may communicate by exchanging messages through the so called ports, which are part of the object interface. Related concepts are protocols, which specify the set of valid messages on each port as well as their direction (i.e. incoming and out coming messages). In this case, the whole behavior of a given class can be specified without any prior knowledge about which objects are going to receive or send messages. According to many authors [Sel94, Szp99] this leads to a more modular design process, on which the reuse of pre-defined components is encouraged.

When communicating using ports, an object (also called component [Szp99] or actor [Sel94]) is not only restricted to dealing with other by way of their interfaces rather than their internals, but is also restricted to dealing with only certain segments of interfaces, as defined by ports. This means that a port exposes a partial set of the receiver methods to the sender, thus allowing the possibility that an object can present different "faces" to the objects with which it interacts by way of different ports. Some methodologies, like ROOM, have stronger protocol policies and demands that binding between ports can only in the so-called conjugated ports. Input and output ports are called conjugated when outgoing messages from a port are accepted as incoming messages in the connected port on the another object. In general, the establishment of such a connection occurs at design time prior to system execution.

Considering the component-oriented communication model [Szy99], each component should define precise input and output interfaces. The component interface defines its access points. These points allow clients of a component, usually components themselves, to access the services provided by the component. Since a component can have multiple interfaces, corresponding to different access points, one can consider that a component interface is logically equivalent to a port. Anyway, it should be pointed out that there is no necessity for the use of protocols, once the component will not explicitly send messages through the interface.

3.2.3 Publisher-Subscriber

This model adopts a content-based producer-consumer oriented style of communication. An object as a producer of a message may spontaneously publish this message. The message can be identified by its content. Other objects interested in this content may subscribe. All objects that have subscribed to a certain content are notified when the respective message is published. Because messages are not routed by address, the communication relation can be determined at run time. The respective local communication subsystems filter the message stream to identify all messages to which local objects have subscribed.

The Publisher/Subscriber communication approach reflects an event-based style of object interaction. A publisher spontaneously sends a message that is going to be delivered to all subscribers that have explicitly requested this individual piece of information. Moreover, a subscriber may subscribe to an entire class of messages offered by the publisher. The subscriber is then notified, whenever a message of a requested class is sent by the publisher. The class is related to the content of a message. Therefore, rather than names or addresses of objects the content is used to route a message to its destination. Conceptually, communication is provided by a single channel, in which all messages are broadcast. To notify the objects when a message of the subscribed class is pushed in the broadcast channel, the communication subsystem has to provide appropriate filtering functions. A publisher does not have to know which will be the receivers of his messages. Vice versa, an object interested in a certain content can extract a message from the space without having to know the publisher. In this way, communication is anonymous and the producing objects and the consuming objects are completely decoupled.

Clearly, many of the above mentioned goals for a communication and interaction scheme are met by the P/S model. However, the mapping of the content-based approach to a physical network may not be straightforward. As described in [KaM99],

depending on the network structure there are different ways to implement the scheme. Roughly, there is a trade-off between efficiency and flexibility. If the properties of the underlying network are completely neglected, the content-based approach requires that every message is examined by every node to decide whether an object on the node has subscribed to the respective content.

Therefore, event channels are introduced in [KaM99] which allow the late binding of message content to addresses. This binding happens at run time just before communicating the first time, i.e. when subscribing to a content class or when publishing the first time. It should be noted that the way this binding is performed is dependent on the underlying network structure. A more detailed review of possibilities can be found in [KaM99].

3.3 Case Study

To highlight the different properties of the interaction schemes in a realistic scenario, the JANUS robotic system has been adopted as a test bench. JANUS is a stationary robotic system consisting of a vision system and two complex arms as manipulators.

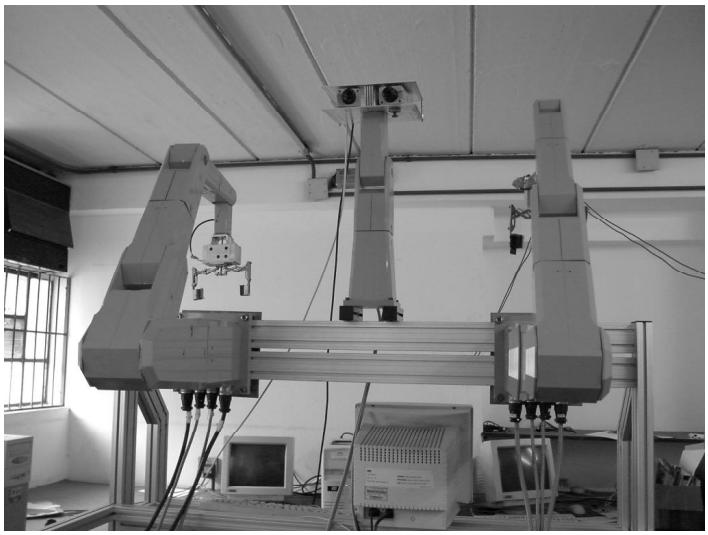


Fig. 2a - Janus

The vision system is mounted on a neck with two degrees of freedom. To manipulate or grap objects, JANUS is equipped with two arms which consist of three segments and eight degrees of freedom each. All joints are monitored by optical encoders which together with the vision system form a complex sensor network. Figure 2a and b depict the Janus system and the proposed control architecture.

The distributed model of control used in this example is based on a multi-agent approach [BoL97]. In this model, the system is hierarchically partitioned in two levels:

(i) a reactive level, where an agent (or object, or component) is associated with each joint, having the sensor and actuator capabilities shown in Figure 2b;

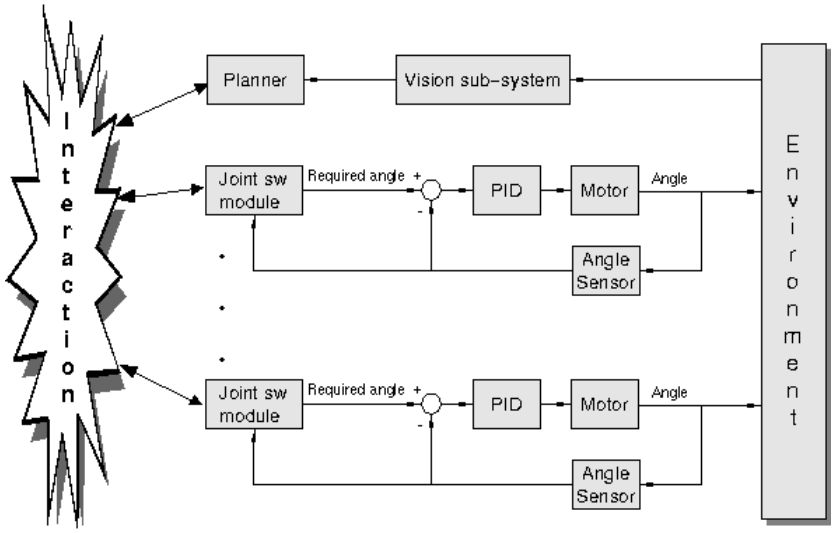


Fig. 2b - Control Architecture

(ii) a planning level, where a planner must communicate with the agents in the reactive level in order to give then directives and goals. For our purposes, a goal is simply a cartesian position to be reached by the effector , eg. grabber, mounted at the end of an arm.

Once a plan is ready, it is communicated to all agents compounding a member, as illustrated in figure 2, and the distributed control algorithm is executed. This algorithm will be briefly explained here for the 2 dimensional case (see Figure 3). The cycle starts at the end-effector related to the joints, which minimizes the angle defined by the lines connecting the respective joint and the effector and the line connecting the joint and the Cartesian coordinates of the goal, as in Figure 3(a).

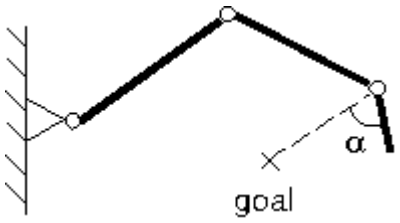


Figure 3a;

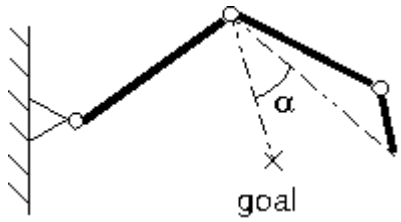


Figure 3b

Once this is done, the new position of the joint is communicated to its most consequent joint, together with a token that must be passed to the neighbor joint. The token circulates through all joints, each one performing the minimization criterion, until the goal is reached. Off course, the algorithm is over simplified here and some constraints may be required, but our focus is on the communication issue.

In order to highlight the differences between the communication strategies presented in the previous section, the JANUS robot system was modeled in three different styles. In the next subsections we show the three different models developed for the JANUS robotic system.

3.3.1 RMI Communication Model

The general components of JANUS robot system are presented in the class diagram in Fig. 4. However, the class diagram does not represent the communication relations

that we want to analyze. For the present example, the communication relations which are necessary during run-time are depicted in the instance diagram of Fig. 5. Because the object has to establish a point-to-point communication with the next consequent joint (for the RMI), a number of 16 connections must be depicted for each arm.

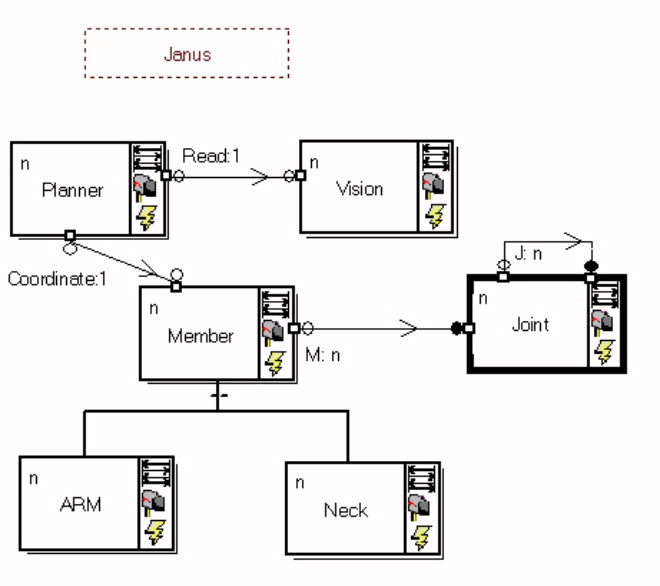


Fig. 4: Object-oriented model for the JANUS Robot System.

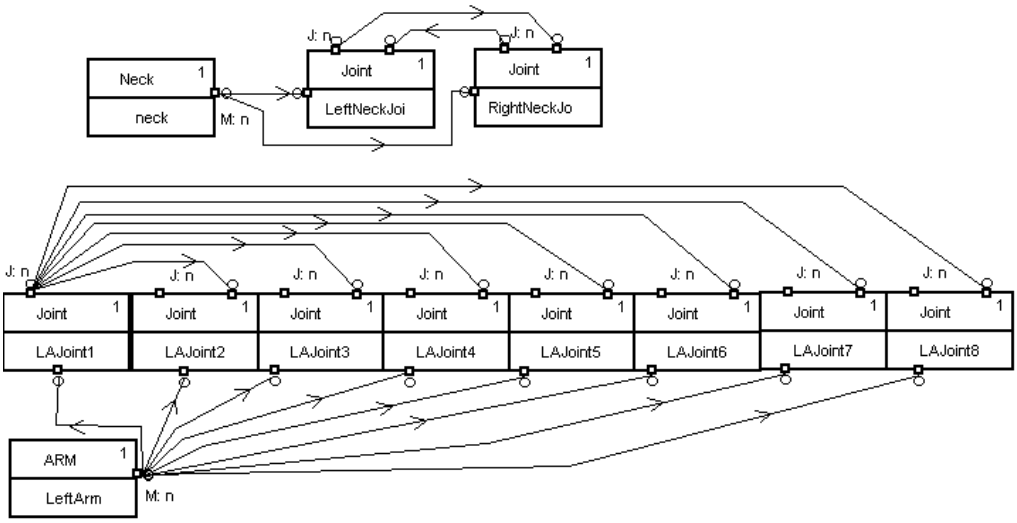


Fig. 5: Instances connections in the RMI communication model

In this communications model; each connection represents a RMI. When executing a step of the control algorithm, each join needs to make a RMI to the next consequent joint. If a joint wants to broadcast its new position to the others joints, it explicitly has to send a RMI to its neighbor joint. In the JANUS approach the arm is moved in steps, and each step requires one move for each joint. As a result 8 RMI , with a similar information and addresses, have to be exchanged during each movement step. This way, a control plane change will lead to sending hundreds of synchronization messages. Hence, the RMI model of communication is not well suited for this scenario, because references are used to identify the communication target, modules

cannot be developed independently. The knowledge of the object (and its interface) to which a communication relation is maintained is required at design time. From the real-time point of view, the temporal properties of the RMI are dependent on the properties of the receiver object.

3.3.2 Port-based Model

A way to better support modular design and active components from the communication point of view is the introduction of ports. A port is an abstraction of a communication channel and is part of the object's interface. Rather than defining a communication relation explicitly in terms of object references, a port reference is a declaration of a component in some higher class definition. Thus, in a way it defines what information is communicated over a port rather than which object is the communication target. That means that the binding of the objects involved in a communication relation are deferred from design time to a later stage of system development, e.g. to the time when the system is configured. From a programmers' point of view, a port clearly removes the problem of programming a broadcast as an explicit sequence of RMIs. Additionally, ports support active behavior of an object. The port-based model for JANUS is presented in figure 5. The runtime configuration from both models, characterized by the instance diagram, have the same connections configurations. Anyway, the main difference between them concerns to the object implementation.

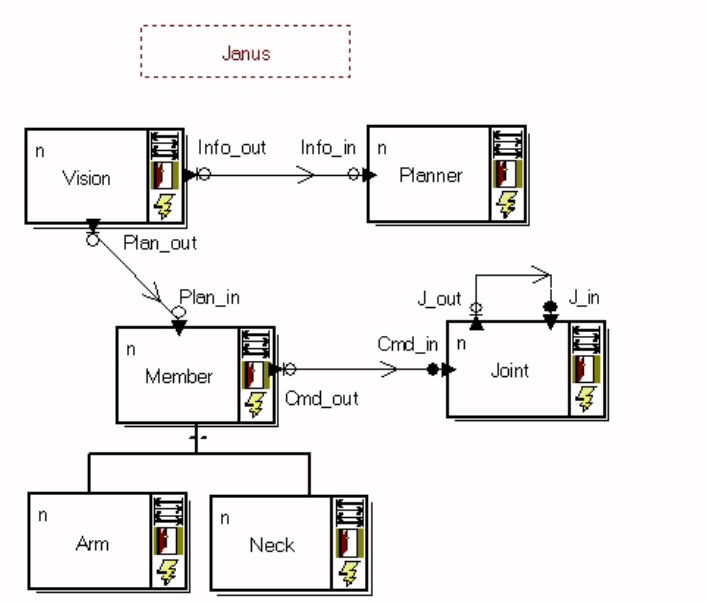


Fig. 6: Port-based model for the JANUS Robot System

In the RMI, the sender is responsible for providing the messages recipient. In the port-based, the sender does not know the receiver, it just writes to an output port, and all connected input ports will receive this message. This way, although not explicitly write by the programmer, both models have the same number of send/received messages.

3.3.3 Event Channel Model

In the Event Channel Model, the event channel is an explicit system component. Therefore the event channel has to be added to the class diagram. Figure 7 shows the class and the instance diagram of the Janus robot using event channels.

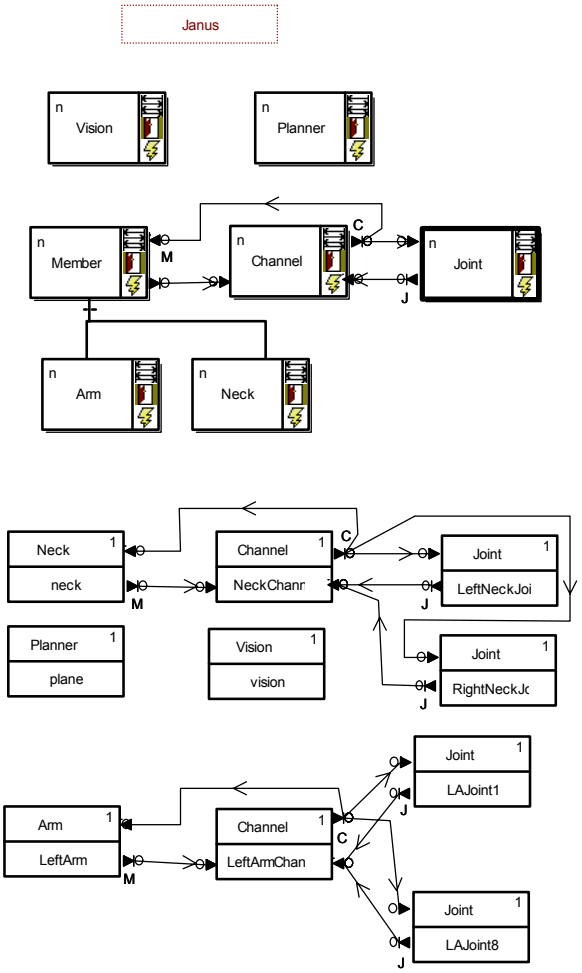


Fig. 7: SIMOO-RT Janus Event Channel Model

The advantages for the system designer to the previous modeling techniques become obvious immediately. There is an event channel "LeftArmChannel" which handles all communication in the left arm of the robot (the right arm is modeled correspondingly). All joints publish their positions in this channel. At the same time, all joints have subscribed to messages handled by this channel. The many-to-many communication relation explicit in the RMI model is now collapsed to a single broadcast channel. As a consequence will have 18 Member to Joints and Joint to Joints connections in the instance diagram. One step movement of the arm represents, at the highest level, just 8 messages. In practice, since the implementation is carried out through an Ethernet network, the number of synchronization messages which need to be modeled for a complete plane move will be one magnitude higher than in the RMI and the Port-based model.

3.3.4 Comparison

In the previous section we compared the different schemes on the design level. Our main goal was to show that broadcast communication patterns that are often needed in control systems are not well supported by the RMI model. Now let us consider modularity, extensibility, and autonomy aspects of the communication models.

As described earlier in the paper, remote message invocation usually couples the transfer of data with a transfer of control thus violating the principle of control autonomy which is crucial in a distributed control system. Additionally, when using remote invocation it is necessary to know which object to address. Thus, the programmer cannot design an object independently from other objects. As a result, any changes in the communication relations will affect a number of objects that may be difficult to trace.

In a port-based communication model, a port just defines outgoing and incoming messages for an object. Thus, the designer of an object does not need to know which other objects are involved in the communication as it is the case when using remote invocation. This supports the important feature of anonymity, i.e. the producer of an information instance does not need to know which are the consumers of this information and vice versa. Anonymity is an additional isolation property that supports modularity and eases modifications. To establish a communication relation between objects, a connection between ports has to be defined in an extra binding step. Usually this binding is performed as part of the configuration when composing the system out of the individual objects. Clearly, this extra step supports modularity, extensibility and modifiability of the system because now the individual objects are not affected or must be modified when changing the communication relations. Concerning autonomy of objects, ports do not force any form of control transfer as it is done in remote invocation.

It should be noted, however, that although ports de-couple the design of a component from the definition of its communication relations, connections between ports are statically configured. This means that during run-time communication relations cannot dynamically be changed, added or removed. Sometimes however, it is necessary to provide uninterrupted control, which requires on-line modifications or extensibility of the system [OPS93]. If this is the case, communication relations have to be changed or added during run-time. As described earlier, the publisher/subscriber communication model has been designed to meet this requirement.

Communication Mechanism	Remote Method Invocation (RMI) [JAV99, VIN99]	Ports [SEL94, SVK97, SZP99]	Event Channels [RGS95, Maf97, HLS97, KaM99]	Content -based Communication [CRW99, OPS93, CaG89, Mor93, KRB99, EGH99]
Basic Comunication Model	Client-Server	Producer-Consumer	Producer-Consumer	Producer-Consumer
Routing mechanism	Client/server name or address	Port name or address	Channel type	Message content
Binding Time	Design Time	Configuration Time	Run Time	No binding
Control Transfer	Yes	No	No	No
Basic Topology	Point-to-point	Point-to-point	Multicast/ Broadcast	Broadcast
Filtering	Sender	Sender	Receiver	Receiver

Table1: Summary of communication characteristics

In its most general form it uses a broadcast to all sites that then have to examine the complete stream of messages to filter those messages which are needed locally. This mechanism avoids any binding but at the price of a substantial load for the filtering function.

The introduction of event channels that bind a message content to a network address overcomes this problem. Event channels are in many respects similar to ports. The difference is that the binding is dynamically performed during run-time when an object wants to communicate the first time. Therefore the binding has not to be specified at any instance during the design or configuration process. Clearly, dynamic binding needs some effort during run-time what is to a certain extent dependent on the underlying network structure. In [RGS95] a distributed set of IPC-demons keep track of the binding between channels and network addresses. In [KaM99] a central broker holds the binding tables. Whenever an object communicates via an event channel the first time, this event channel broker is involved in resolving the channel to address binding. It should be noted that the binding has only be performed once and does not necessarily happen in the real-time loop or the “steady state path”[RGS95]. The binding could e.g. be performed in a boot phase for a new component when it is integrated in the system. Table 1 summarises the properties of the schemes discussed here.

3.4 Conclusions

The paper has compared different strategies for interacting and communicating distributed real-time objects. From the case study one can clearly observe that the remote method invocation concept, the most frequently adopted communication model in distributed object-oriented systems has several drawbacks when compared to other approaches. Not only the fact that communication binding must be done explicitly at design time , thus decreasing the reusability of the same class in different contexts, the adopted point-to-point communication leads to ineffective way of communication in distributed control systems.

Currently, time measurements of the implement alternative solutions for the Janus case study are being performed. They will allow a comparison of the obtained real-time properties, such as execution time, cyclical activation jitter, and specially the differences in the temporal behavior caused by the overhead imposed by the different communication schemes.

3.5 References

[BoL97] P. Bohner, R. Lüppen, "Redundant Manipulator Control Based on Multi-Agents", 3rd IFAC Symp. on Intelligent Components and Instruments for Control Applications - SICICA' 97, Anecy, France, pp 357-362, June, 1997.

[CaG89] N. Carriero, D. Gelernter: "Linda in Context", Commun. of the ACM, 32, 4, April 1989, pp 444-458.

[CRW99] A. Carzaniga, D.S. Rosenblum, A.L.Wolf:”Achieving Scalability and Expressiveness in an Internet Scale Event Notification Service”, 1999

[EGH99] D. Estrin, R Govindan, J. Heidemann: "Scalable coordination in sensor networks”, Proc. ACM/IEEE MobiCom, 1999

[HLS97] T.H. Harrison, D.L. Levine, D.C. Schmidt: "The Design and performance of a Real-time CORBA Event Service", Proc. of the 12th Ann. Conference on Object-oriented Programming, Systems, Languages and Applications, OOPSLA, Atlanta, USA, 1997.

[JAV99] Javasoft. Java RMI Enhancements since JDK 1.2. www.javasoft.com/products/ jdk/1.2/docs/guide/rmi/index.html, 1999.

[KaM99] J. Kaiser, M. Mock: "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)", Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99), Saint-Malo, France, May 1999.

[KRB99] J. Kulik, W. Rabiner, H. Balakrishnan;"Adaptive Protocols for Information dissemination in wireless sensor networks", Proc. of the ACM/IEEE MobiCom, 1999

[Maf97] S. Maffeis: "iBus - The Java Intranet Software Bus", Olsen&Associates, www.olsen.ch, 1997.

[Mor93] K. Mori. Autonomous decentralized Systems: Concepts, Data Field Architectures, and Future Trends, Int. Conference on Autonomous Decentralized Systems (ISADS93), 1993.

[OPS93] B. Oki, M. Pfluegl, A. Seigel, D. Skeen: "The information Bus®- An Architecture for Extensible Distributed Systems", 14th ACM Symposium on Operating System Principles, Asheville, NC, Dec 1993,pp.58-68.

[Pointc] Pointcast Inc., The Pointcast Network, www.pointcast.com .

[RGS95] R. Rajkumar, M. Gagliardi, L Sha: " The Real-Time Publisher/Subscribe Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation", IEEE Real-time Technology and Applications Symposium, June 1995.

[Sel94] Selic, B., Gullekson, B., Ward, P. Real Time Object Oriented Modelling. John Wiley and Sons, Inc., 1994.

[SVK97] D.B. Stewart, R.A. Volpe, P.K. Khosla:"Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects", IEEE TC on Software Engineering, Vol.23, No.12, December 1997

[Szy99] Szyperski, Clemens. Component Software: Beyond Object-Oriented Programming. Addison-Wesley. Londres. 1999.

[VIN99] Vinoski, Steve. Advanced CORBA Programming with C++. Addison-Wesley, Reading, Massachusetts, 1999.

4 Cooperation through the Environment: Stigmergy in CORTEX

Greg Biegel
Distributed Systems Group
Trinity College Dublin

This chapter decribes a way of coordinating sentient objects through the environment. The technical term "stigmergy" was coined by a biologist which used it to describe the co-ordination of populations of insects without direct communication between the individuals. The concept is discussed in realtion with sentient objects which are able to exploit their sensoric and actoric real-world interfaces to support a robust and efficient coordination mechanism for large numbers of interacting mobile entities.

Stigmergy in CORTEX[6]

Gregory Biegel
Distributed Systems Group
Trinity College Dublin

Greg.Biegel@cs.tcd.ie

4.1 Introduction

Stigmergy, or coordination of actions through the environment, was first observed in colonies of insects [1]. Stigmergy as a coordination mechanism is characterised by a lack of planning using explicit communication between entities, a fact that makes it extremely flexible and robust in large systems. Stigmergy is based upon sensing and sampling the environment and responding to it. Sentient objects as defined in CORTEX, are software components that sense their environment via sensors and alter it via actuators. We propose stigmergy as a possible coordination mechanism for large networks of real time sentient objects.

4.2 Stigmergy Defined

The term stigmergy was first introduced by Pierre Paul Grassé, a French entomologist, in 1959. He used the term to describe the coordination of the activities of ants in carrying out complex activities, such as nest building, without direct communication amongst themselves. The Greek origins of the word itself translate to mean 'incitement to work by the products of work themselves' [1].
Stigmergy may be observed in the physical world when termites' nest building activities are influenced by certain configurations of the nest itself (Grassé cited in [2]). The alteration of the environment is not always visible, and stigmergy amongst ants relies on the creation of a dissipative field by the spreading of chemical substances, known as pheromones, in the environment. Pheromones automatically guide ants to certain activities and are spread as the result of activities. In this way the results of actions performed direct future actions and behaviour.
Stigmergy in ant colonies allows the development of complex coordinated behaviour through the actions of single ants. The gathering of food is a good example. If a lone ant finds some food, it will collect it and when returning to the nest, will leave a pheromone trail from the source of the food. The pheromone will influence the behaviour of other ants to go towards the same source. If they too find food, their behaviour is the same, and thus the pheromone trail is reinforced (positive feedback). When the food is exhausted, ants will no longer go to the source and the pheromone trail will disappear.
It is evident that stigmergy describes a form of asynchronous interaction and information interchange between entities mediated by an 'active' environment [3].

The active environment for ant colonies is the air containing pheromones and sensed by individual ants. Most importantly in stigmergic coordination, there is no **direct** communication between entities requiring coordination, an entity simply signals its

6

intentions or actions to the environment. Other entities sampling the environment detect these signals and act accordingly.

4.3 Applications of stigmergy

Stigmergy has seen application in a range of domains such as network routing [4], load balancing in telecommunication networks [5] and especially in the field of robotics and more specifically cooperating autonomous robots. Holland et al. describe the use of stigmergy in a collection of robots exhibiting simple behaviour, to achieve sorting of objects [2]. Although not explicitly using the term 'stigmergy', Werger [6] describes stigmergic coordination in the team behaviour of a robotic soccer team where coordination is achieved solely through the environment, without explicit communication between members.

4.4 Components of a stigmergic system

All systems in which stigmergy plays a role have three components that collectively enable stigmergic coordination [3]. *Entities* are actors amongst which coordination is required. The *Environment* represents the physical environment in which entities exist and through which an *Information Carrier*, such as a pheromone in ants, distributes information. These components are illustrated in Figure 1. The Environment also defines a distribution mechanism for information in the environment.

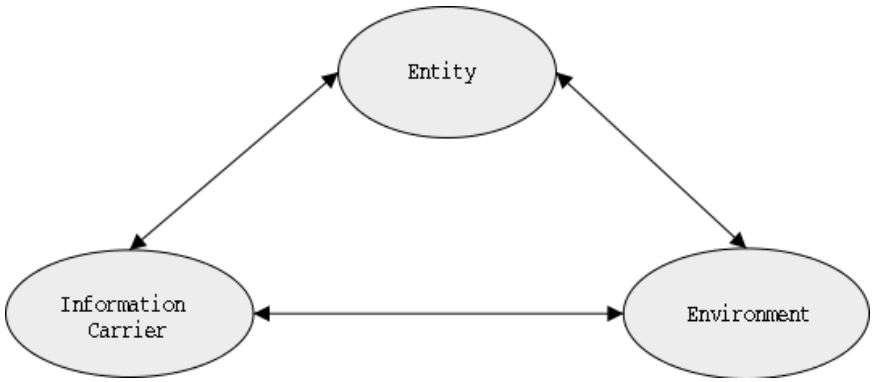


Figure1: Elements of a stigmergic system (adapted from [3])

4.5 Stigmergic coordination in CORTEX

CORTEX is concerned with large-scale networks of mobile sentient objects co-operating together to achieve some application goal. CORTEX explicitly recognises communication through the physical environment and defines sentient objects as having the ability to alter their environment. In CORTEX, the environment constitutes an interaction and communication channel and is in the control and awareness loop of the objects [7].

The size and scale of envisaged CORTEX networks preclude global knowledge and control within a network of sentient objects. Coordination is required amongst sentient objects in order to accomplish system goals, but the size of the networks means that centralised coordination is not a good choice due to considerations such as the presence of a single point of failure and the inability of the centralised model to scale well. A model of coordination based upon local knowledge and decision-making, such as that offered by stigmergy, is more appropriate. As has been seen, in

this model coordination occurs through local knowledge and action, whilst having a global overall effect. Furthermore, stigmergy recognises the role of the physical environment as a communication channel, a core concept in CORTEX.
Context awareness plays an important role in stigmergy through the maintenance of a model of the environment, which is both altered by, and influences behaviour. The importance of a model of the environment in stigmergy is explained further in Section 3.1.2.

4.6 Environment and Architecture

For stigmergic coordination to be used in CORTEX, the three major components of any stigmergic system described in Section 2.1 need to be mapped in the CORTEX architecture.

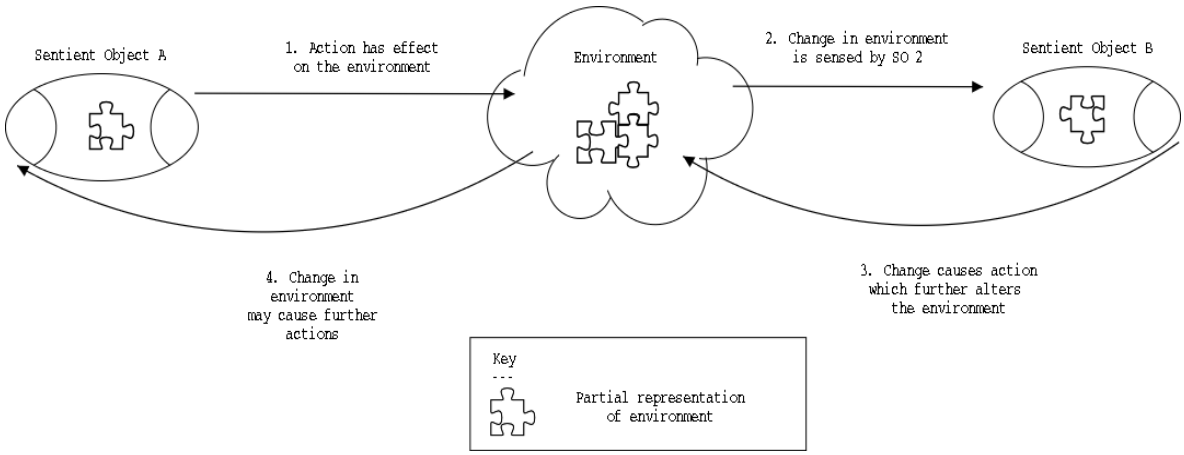


Figure 2: Stigmergic coordination in sentient objects

4.6.1 Entity

The entities amongst which coordination has to be achieved in CORTEX are sentient objects [8]. Sentient objects are eminently suited to stigmergic coordination due to their characteristic sensing and sampling of their environment, and potential to both change their environment and respond to changes in their environment.

4.6.2 Environment

The *Environment* component in CORTEX is realised through context awareness in which each sentient object maintains a view of its environment. The view of the environment includes both the *internal* environment, which includes information such as group membership and network delays, and the *external* environment, which includes information about other sentient objects. This model reflects a programmer specified view of the real world and is distributed in both space and time, across the sentient objects in the system. Stigmergic coordination in software systems requires that sentient objects maintain a model of the real world (environment) that captures changes in the real world. This allows the behaviour of a sentient object to be influenced by changes made to the environment and is closely allied to the concept of Context Based Reasoning [9]. The view of the environment as maintained in a context

aware object may be suitable for use in stigmergic coordination, or may need to be adapted to provide a useful model.

4.6.3 Information Carrier

The Information Carrier component is the only component which has not been explicitly considered in CORTEX at this point. The current definition of sentient objects describes two types of events, *real world* events and *software* events [8]. Real world events are those events produced in the physical environment, such as a change in light intensity, or temperature. Software events are events produced by a software entity and may describe a real world event. Biologically speaking, pheromones are real time events, being a chemical change in the air. Software events in CORTEX appear to be very similar to pheromones in that they carry information, and are effectively dispersed through the networking environment. Real world events may also act as pheromones when their production is under the control of a sentient object. For example, a sound is a real world event and if produced from a speaker under the control of a sentient object, may act as an information carrier if the sensing of the sound is meaningful in some way to another sentient object.
So both software and real world events as defined in CORTEX may act as information carriers in stigmergic coordination. The use of real world events as an information carrier is closer to the original definition of stigmergy and would require the use of actuators that produced real world events.
The basis of stigmergic coordination amongst sentient objects is illustrated in Figure 2. Sentient object A performs an action that alters the environment in some way (through production of an event). This action is not explicitly communicated to object B, but object B senses the change to the environment through a change in its internal representation of the environment. This causes object B to perform an action that may in turn alter the environment. The state of the environment is the common stimulus to both sentient objects.

4.7 Relation to the event service

Events are the natural choice of information carrier for a stigmergic coordination system in CORTEX. CORTEX already defines an event-based communication model, and as an anonymous generative communication paradigm, events fit well with the stigmergic coordination model where coordination does not involve point-to-point communication.
At first investigation, stigmergic coordination does not have any special requirements from the event service. Any event type may be considered an information carrier, or pheromone. The interpretation of these events in the context of overall coordination of the system poses the greatest challenge in the development of a stigmergic coordination system.
In terms of real world events, which are less under the control of a sentient object than are software events, an actuator creating information carrying real world events (a kind of artificial pheromone secreting gland) is a possibility.

4.8 Why stigmergic coordination?

When considering the CORTEX architecture of a large number of mobile objects, communication between objects is expensive, since it is potentially very expensive to locate those objects interested in a particular message. A coordination strategy which

would involve planning between a number of objects would need to deal with location, membership and agreement issues none of which are easily solved.
Stigmergic coordination does not require any explicit planning of the coordination between objects, each object makes changes to the environment and acts upon changes it senses being made to the environment. Stigmergic coordination does not require centralised control and is thus a robust coordination mechanism. Stigmergic coordination systems have the ability to handle highly dynamic situations. The arrival or departure of objects does not require any alteration of planning or additional computation, each object simply continues as before. Stigmergic coordination allows the development of complex interaction patterns from relatively simple local actions, and may be achieved in real time since it does not require any sort of signalling [10].
Another advantage is that stigmergic coordination via real world events in CORTEX is not liable to delays introduced by network or computational infrastructure and as a result is promising with regard to the real-time element of CORTEX. Finally, stigmergic coordination is efficient as a result of not having to plan coordination through communication with all participating objects.

4.9 Disadvantages of stigmergic coordination

Stigmergic coordination is not suited to all problem domains and is better suited to specific domains such as attraction to specific locations, or attraction to move in a specific direction. Stigmergic coordination may not be suitable to all CORTEX application scenarios, but shows promise in application scenarios such as sentient traffic systems, or sentient air traffic control, where dynamic routing of entities is required.
Stigmergic coordination requires a fairly high degree of redundancy that is a large number of entities. CORTEX defines networks of very large numbers of sentient objects, so a degree of redundancy exists in CORTEX networks.
Procedures for developing stigmergic coordination amongst objects are not defined and it is here where the greatest challenge lies to the use of stigmergic coordination. The specifications of local actions that drive the system towards its overall goal pose a great challenge to the systems developer.

4.10 Conclusions

In this paper we have presented stigmergy as a potential form of coordination for sentient objects in CORTEX. Stigmergy refers to the global coordination of the actions of a group of entities through local interactions with the environment, as observed in insect colonies.
Sentient objects sense their environment through sensors and alter it via actuators and recognise communication through the environment. It is proposed that stigmergy provides an efficient and robust mechanism for coordination amongst large numbers of highly mobile sentient objects, whilst recognising that stigmergy is not necessarily appropriate for all sentient object application scenarios.

4.11 References

[1] Adam Russell *Coordination of group behaviour through stigmergy: possible implications for theories of communication* Sixth European Conference on Artificial Life (ECAL) January 2001

[2] Holland Owen Holland, Chris Melhuish *Stigmergy, Self-Organisation, and Sorting in Collective Robotics* Artificial Life 5: 1999 pp. 173-202

[3] Paul Valckenaers, Martin Kollingbaum, Hendrik Van Brussel, Olaf Bochman, C. Zamfirescu *The Design of Multi-Agent Coordination and Control Systems using Stigmergy*
In Proceedings of the IWES'01 Conference, Slovenia 2001

[4] Gianni Di Caro, Marco Dorigo *AntNet: Distributed Stigmergetic Control for Communications Networks* Journal of Artificial Intelligence Research 9 1998

[5] Ruud Schoonderwoerd, Owen Holland, Janet Bruten, Leon Rothkrantz *Ants for load balancing in telecommunications networks* Hewlett Packard Labs Technical Report HPL-96-35

[6] Barry Brian Werger, *Cooperation without deliberation* Artificial Intelligence Vol. 110 N0. 2 1999

[7] Project Partners *CORTEX Technical Annex* October 4, 2000

[8] Adrian Fitzpatrick *Sentient Object Definition* Trinity College Dublin 2002

[9] Fernando G. Gonzalez, Grejs Patric, Avelino J. Gonzalez *Autonomous Automobile behaviour through Context-based Reasoning*, University of Central Florida, `http://isl.engr.ucf.edu/publications/CxBR.html`

[10] Stan Franklin Coordination without Communication University of Memphis, `http://www.msci.memphis.edu/~franklin/coord.html`

5 Dynamic dependable Quality of Service Adaptation

António Casimiro
Paulo Veríssimo
FC/UL*

The definition of the CORTEX interaction model encompasses several aspects, such as the definition of suitable communication abstractions, the treatment of co-ordination and predictability aspects, vis-à-vis the need for autonomy and anonymity, and the provision of support for non-functional needs of sentient applications. The work has been structured around a task (Task 2.2) where context and environment awareness related issues are studied. For this task, the present deliverable provides some results concerning the aspect of guaranteeing temporal properties of interactions. In concrete, it provides a paper that discusses how to achieve *dependable QoS adaptation*, in the context of a partial synchrony model, namely the Timely Computing Base Model (*see* deliverable WP3-D4).

The need to use QoS adaptation techniques, which is not particularly new and has been studied in the context of adaptive systems, derives from the need to satisfy non-functional requirements of sentient objects. However, the possibility of adapting in a dependable way and to provide a certain degree of predictability, which is essential for CORTEX applications, has not been fully understood. Dependable adaptation requires a form of context and environment awareness, whose properties must rely on accurate and reliable monitoring information delivered by appropriate middleware constructs. Dependable adaptation also encompasses notions fundamental for co-ordination and cooperation among sentient objects, such as agreement on monitoring information, and timeliness of information delivery. Finally, since the environments we are dealing in CORTEX are typically unpredictable or unreliable, the problem of dependable adaptation must be equated in an adequate framework, where these partial synchrony properties are taken into account. We use the Timely Computing Base model as our basic framework, and therefore the paper also describes its basic properties and services.

* Faculdade de Ciências da Universidade de Lisboa. Bloco C5, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-2000-26031 (CORTEX), and by the FCT, through the Large-Scale Informatic Systems Laboratoty (LASIGE) and projects Praxis/P/EEI/12160/1998 (MICRA) and Praxis/P/EEI/14187/1998 (DEAR-COTS).

Using the Timely Computing Base for Dependable QoS Adaptation

António Casimiro
Paulo Veríssimo
FC/UL*

Abstract

In open and heterogeneous environments, where an unpredictable number of applications compete for a limited amount of resources, executions can be affected by also unpredictable delays, which may not even be bounded. Since many of these applications have timeliness requirements, they can only be implemented if they are able to adapt to the existing conditions. Adaptation can be done by several ways, taking into account many different factors, but an obvious factor of success is knowing what they have to adapt to. In this paper we present a novel approach, called Dependable QoS adaptation*, which can only be achieved if the environment is accurately and reliably observed.*

Dependable QoS adaptation is based on the Timely Computing Base (TCB) *model. The TCB model is a partial synchrony model that adequately characterizes environments of uncertain synchrony and allows, at the same time, the specification and verification of timeliness requirements. We introduce the* coverage stability *property and show that adaptive applications can use the TCB to dependably adapt and enjoy this property. We describe the characteristics and the interface of a* QoS coverage service *and discuss its implementation details.*

5.1 Introduction

An increasing number of applications with timeliness or real-time requirements is being used in open, unpredictable or unreliable environments, like the internet. This is the case of multimedia applications, e-commerce or transaction based applications and applications for remote process control. For a systems' architect this may appear to be a contradiction, since it is well known that no real-time guarantees can be provided using intrinsically asynchronous platforms or environments. What happens in practice is that many of these applications are simply best-effort applications, or they are designed artificially assuming a synchronous system model. In the latter, the apparent guaranteed (synchronous) behaviour is only achievable until the first occurrence of a *timing failure*.

Any systematic approach to the problem of implementing timeliness requirements in environments with unpredictable behaviour has to take into account the effects of

* Faculdade de Ciências da Universidade de Lisboa. Bloco C5, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-2000-26031 (CORTEX), and by the FCT, through the Large-Scale Informatic Systems Laboratoty (LASIGE) and projects Praxis/P/EEI/12160/1998 (MICRA) and Praxis/P/EEI/14187/1998 (DEAR-COTS).

timing failures on the correctness of applications. However, this is not what happens with most of the known system models. For instance, in asynchronous, or time-free[16] systems, there is not even a notion of time and in synchronous systems the fault model includes crash and omission failures, but no timing failures. The timed asynchronous system model [11] allows the definition of timed services and "knows" that timing or performance failures can occur. But its basic asynchronous nature does not allow timing failures to be detected within known time bounds. So their effects can only be partially handled.

The lack of a generic model able to deal with the partial synchrony problem in a systematic way was one of the reasons that motivated our work around the definition of a new model, which we called the **Timely Computing Base (TCB)** model [23]. It assumes that systems, however asynchronous they may be, and whatever their scale, can rely on services provided by a special module, the TCB, which is timely, that is, synchronous. Under the TCB framework we define different classes of applications according to the properties that they enjoy, and we explain how to handle the effects of timing failures, with the help of TCB services, for each of these application classes (or combinations thereof).

In this paper we concentrate on the particular effect of *decreased coverage* resulting from timing failures, and show that there is a class of QoS adaptive applications, to which we refer as *time-elastic* applications, that may benefit from the TCB to avoid the decreased coverage problem. An important aspect of our work is that adaptation to environment changes relies on rigorous observations and mathematical analysis, which allows to dependably adapt the QoS (expressed as timing variables) to maintain the coverage of timeliness assumptions. We describe the QoS coverage service as an entity that can indeed be used by applications to dependably adapt.

The rest of this paper is organized as follows. Section 2 presents a brief overview of related work. Then, in section 3 we briefly present the key aspects of the TCB model. The problem of dependable adaptation is discussed in section 4, focusing on the role of the TCB in the global system architecture. Section 5 presents the QoS coverage service and in section 6 we discuss a few implementation issues. Finally, in section 7 we present our conclusions.

5.2 Related Work

The provision of quality of service (QoS) guarantees in open environments, such as the Internet, is currently an active field of research. In fact, although there is a lot of work dealing with the problem of QoS provision in environments where resources are known and can be controlled [25,21,27,17], no systematic solution has been proposed for environments where there is no knowledge about the amount of available resources. As we know, the Timely Computing Base (TCB) model is the only system model that characterizes these kind of unpredictable environments in a generic way, and is thus adequate to derive solutions for applications with QoS needs.

Most of the works that deal with QoS provision assume that resource reservation is possible. They address several facets of the problem and propose very diverse solutions. For instance, they propose to use benefit functions specified by the application as a way to optimise resource management [19], they describe middleware

architectures to deal with heterogeneous environments [21], they propose control-based solutions [18] or they use resource brokers to manage system resources [27]. A comprehensive survey about end-to-end QoS architectures can be found in [4]. The IntServ [6] and DiffServ [5] architectures were proposed to specifically address the problem of handling QoS requirements and differentiated service guarantees in the Internet. Detailed discussions about multimedia applications over the Internet can be found in [14] and [26].

Unlike the majority of the work dealing with QoS provision, we are concerned with environments where resource reservation and QoS enforcement may not be possible. Obviously, not all applications can be implemented on these environments. They need to be *adaptive* or more particularly *time-elastic*, that is, they must be able to adapt their timing expectations to the actual conditions of the environment, possibly sacrificing the quality of other (non time-related) parameters. The success of an adaptive system has to do essentially with two factors: 1) the monitoring framework, which dictates the accuracy of the observations that drive adaptation and 2) the adaptation framework, which determines how the adaptation will be realized.

Monitoring of local resources and processes is widely used [19,17], but it does not provide a global view of the environment. Some works propose adaptation based on network monitoring and on information exchange among hosts (using specific protocols like RTP [7] or estimating delays [8]) but they do not reason in terms of the *confidence about the observations*, which is essential for dependable adaptation. In contrast, we focus on providing a global view of the environment (consistent to all participating entities) and on ensuring the correctness of that view.

Relatively to adaptation strategies, we mention the work in [1] that proposes adaptation among a fixed number of accepted QoS levels, and the work in [18], that uses control theory to enhance adaptation decisions and fuzzy logic to map adaptation values into application-specific control actions. However, since they have no *dependability* concerns relative to the adaptation, neither they reason in terms of a *generic model of partial synchrony* for the distributed system, we believe our work can complement them. These dependability concerns appear on the AQuA architecture [12], which provides adaptation mechanisms to respond to system faults and to maintain certain (dependability related) QoS levels. But our dependability concerns are related with the adaptation mechanism itself, and not particularly with application dependability requirements.

The study of partial synchrony models has been the subject of some previous work, which have also addressed the problem of QoS adaptation. Cristian and Fetzer have proposed a methodology based on the timed-asynchronous model [11] to design adaptive real-time applications [15]. The quasi-synchronous model [22], developed by one of the present authors, has provided the framework to address the problem of achieving group communication in the presence of timing failures [3]. These works were in general precursors of the TCB model [24], which provides a more generic framework to address asymmetries (both in space and time) of the system synchrony and is therefore adequate to handle a vast range of problems.

5.3 Timely Computing

The Timely Computing Base model provides the framework for the work presented in this paper. However, due to space limitations, this paper just introduces the services and interface provided by a TCB module. All detailed explanations about these and other issues concerning the TCB can be found in other documents, namely in [23] and [24].

A system with a Timely Computing Base is divided into two well-defined parts: a *payload* and a *control* part. The generic or *payload* part prefigures what is normally 'the system' in homogeneous architectures. It exists over a payload network and is where applications run and communicate. In particular, all middleware services dedicated to QoS provisioning, monitoring or management are constructed in the payload part of the system. The *control* part is made of local TCB modules, interconnected by some form of medium, the *control* network. The payload part can have any degree of synchronism, and the control part (the TCB) is assumed to be a synchronous component. A discussion about TCB implementation issues can be found in [9].

A TCB can be turned into an oracle providing time-related services to applications or middleware components. To accomplish this, a set of minimal services has to be defined, as well as a payload-to-TCB interface.

In order to keep the TCB simple, the services defined are only those essential to satisfy a wide range of applications with timeliness requirements. Therefore, the TCB provides a **Duration Measurement** service with the ability to measure distributed durations with bounded accuracy; a **Timing Failure Detection** service able to completely and accurately detect timing failures; and a **Timely Execution** service with the ability to execute well-defined functions in bounded time.

Besides defining essential services to be provided by the TCB, it is very important to provide a programming interface to allow potentially asynchronous applications to dialogue with a synchronous component. One of the most important problems is that the latency of service invocation, as well as the latency of service replies, may not be bounded. So it is not possible to relate (in a time line) events occurring in one side with events occurring in the other. The interface summarized in Table 1 (with a slightly more versatile `waitInfo()` function than the one presented in [24]) makes a bridge between a synchronous environment and a potentially asynchronous one. A description of the several functions and some examples of how to use them are presented in [24].

5.4 Dependable QoS Adaptation

Given the TCB model with its services and interfaces, which we consider the essential ones to address timeliness issues in unpredictable environments, this section analyses the implications of using this model to construct QoS architectures.

Quality of Service can have different meanings that depend essentially on the application. This is why the definition of a completely generic model for specifying QoS needs is perhaps an impossible task: it is usually necessary to use mapping mechanisms to translate user level QoS requirements into system level ones. In the present work we assume that it is possible to define mapping mechanisms, so that QoS requirements of different applications can be specified in terms of timing variables, which are the variables of interest in the TCB model.

Duration Measurement

```
timestamp ← getTimestamp ()
id ← startMeasurement (start_ts)
end_ts,duration ← stopMeasurement (id)
```

Timely Execution

```
end_ts ← exec (start_ts, wait, exec_dur, f)
```

Timing Failure Detection

```
id ← startLocal (start_ts, spec, handler)
end_ts,duration,faulty ← endLocal(id)
id ← send (send_ts, spec, handler)
id,deliv_ts ← receive ()
id, dur1,faulty1 Ö durn,faultyn ← waitInfo()
```

Table 1: Summary of the API.

The TCB model provides a generic framework to deal with synchronism problems and to provide certain safety and liveness properties in the time domain to applications. In this sense, there is a potential for this model to be used as a base model for the development of some application classes. Adaptable applications with QoS requirements is one of those classes.

Typically, QoS adaptable applications are realized on top of a QoS framework. This framework is defined by a compound of QoS mechanisms which altogether characterize its ability to deal with application QoS requirements. There are three basic categories of QoS mechanisms: QoS provision, QoS control and QoS management mechanisms. If we want to use the TCB as a basic model to serve the design of some end-to-end QoS architecture, we must investigate the benefits that it might bring for the implementation of these QoS mechanisms. In particular, and since the TCB model provides a set of services to applications, we must find answers for the following questions: *Which QoS mechanisms can be improved by using TCB services? How can TCB services be used to improve QoS mechanisms?* The rest of the section will essentially focus on the discussion of these questions.

5.4.1 QoS Mechanisms under the TCB Framework

Before all, recall that the TCB does not restrict the properties of the payload part of the system, which allows for any QoS architecture to be implemented in a TCB based

system. In fact, we stress that a TCB is just a small component that, mostly because of its small size and simplicity, can be constructed with more synchronous properties (or can be more dependably synchronous) than the rest of the system. Therefore, simply imagine that a TCB can be plugged into any existing system and be used as an oracle that provides a few basic services. The question is whether these services can be used **to improve** the existing QoS mechanisms. Let us first take a look on these mechanisms, closely following the systematisation presented in [4]).

The first category of QoS mechanisms, QoS provision, allows applications to use low level services, such as communication services, and to specify desired levels of QoS. QoS provision includes the following: a) QoS mapping – to translate application level notation to system level specifications; b) Admission testing – to verify that there exist enough available resources to admit the requested QoS level; c) End-to-end reservation – to reserve resources and prevent resource outage during execution.

The second category consists of QoS control mechanisms, which serve to regulate and control the way in which resources are used and shared by the several flows (traffic or execution flows) during run-time. They include several mechanisms, namely a) Flow shaping – to regulate usage according to some formal representation (for instance, a statistical one); b) Flow scheduling – to manage the several flows in an integrated way; c) Flow policing – to verify that the contract is being adhered to by applications; d) Flow control – to guarantee resource availability, either using open loop schemes (with advanced resource reservation) or closed loop schemes (with flow adaptation based on feed-back information); e) Flow synchronization – to account for precise interactions (e.g. multimedia interactions) of different flows.

Finally, the QoS management category includes the following mechanisms: a) QoS monitoring – to observe QoS levels achieved by lower layers; b) QoS maintenance – to perform fine adaptation of resource usage to maintain QoS levels; c) QoS degradation – to deliver QoS indication to applications when QoS cannot be sustained; d) QoS availability – to allow applications to specify bounds for QoS parameters; e) QoS scalability – which comprises QoS filtering, to allow the manipulation of (traffic) flows, and QoS adaptation, to scale flows at the end system in order to respond to fluctuations in end-to-end QoS.

Now we must recall that TCB services can be used, in a general sense, to help applications observe their timeliness (using the duration measurement service), execute short real-time operations (with the timely execution service) or timely detect timing failures and react upon their detection within given time bounds. We must therefore look for QoS mechanisms that rely directly or indirectly on time, which are the ones that might be improved using these services.

Of the QoS provision mechanisms, QoS mapping and end-to-end reservation only perform logical operations and do not need timeliness information to achieve their goals. Clearly, they are not eligible mechanisms for possible improvements. On the other hand, admission testing mechanisms need to compare available resources with requested ones, and so may possibly use the TCB to have the knowledge of available resources, measured in terms of timeliness. Relatively to QoS control mechanisms, the one that most clearly can use a TCB, in particular its duration measurement service, is the closed loop flow control mechanism. The basic idea is to use the

information about distributed durations (relative to end-to-end transmission of data) to derive conclusions about the correct control actions to take. Finally, the QoS management mechanisms that deal with observing the behaviour of the environment (or lower level services) can also use information about distributed durations to measure the QoS level (in terms of timeliness) that is being provided in a given moment.

The fundamental conclusion is that a TCB can essentially be used to improve the mechanisms that need information about the environment. This is clearly the case of mechanisms such as the flow control or the QoS monitoring, but can also be the case of other mechanisms, such as the QoS maintenance, which by using the output of a possibly improved monitoring can also employ finer grain techniques to achieve better results.

5.4.2 Improving QoS Mechanisms

We have just seen that a TCB can eventually be useful to construct or improve any QoS mechanism that needs information about the environment. Since this information is crucial, among other things, to take correct adaptation decisions, we argue that by using a framework which explicitly addresses the unpredictable nature of the environment in terms of timeliness, it is possible to adapt applications in a dependable manner, based on the timely observation of the environment.

To demonstrate our point of view, and having in mind that timeliness is the fundamental property in this context, we take a constructive approach that consists in analysing why systems fail in the presence of uncertain timeliness, and deriving sufficient conditions to solve the problems encountered, based on the behaviour of applications and on the properties of the TCB.

5.4.2.1 Effects of Timing Failures

We assume that an application is a computation in general and that any application is defined by a set of safety and timeliness properties P_A. In the absence of timing failures, the system executes correctly. When timing failures occur, it is generally assumed that the effect is **unexpected delay**. However, we observe two additional effects, which produce different pathologies in the system, by the way they affect high-level system properties: **decreased coverage** and **contamination**. We define and discuss below the effects that are relevant for the current work. The effect of contamination is discussed in detail in [23].

Unexpected Delay

The immediate effect of timing failures is unexpected **delay**, defined as the violation of a timeliness property. That can sometimes be accepted, if applications are prepared to work correctly under increased delay expectations (e.g. mission-critical or soft real-time systems).

Uncoverage

When we make assumptions about the prevention of timing failures, we have in mind a certain coverage, which is the correspondence between system timeliness assumptions and what the environment can guarantee. We define **assumed coverage** P_P of a property P as the assumed probability of the property holding over an interval of reference. This coverage is necessarily very high for hard real-time systems, and maybe somewhat relaxed for any other real-time system, like mission-critical or soft real-time systems. However, in an environment with uncertain timeliness, the actual coverage varies during system life. Each time the environment conditions degrade to states worse than assumed, the coverage incrementally decreases, that is, the probability of timing failure increases. On the other hand, if the coverage is better than assumed, we are not taking full advantage from what the environment offers. This is a generic problem for any class of system relying on the assumption/coverage binomial [20]: if coverage of failure mode assumptions does not stay stable, a fault-tolerant design based on those assumptions will be impaired, because dependability properties, such as reliability, may be violated, or one does not take full advantage of the system. A sufficient condition for that not to happen, expressed by the *Coverage Stability* property, consists in ensuring that coverage stays close to the assumed value, over an interval of mission.

In [23] we have already presented formal proofs to show that the TCB can help applications to secure coverage stability despite the uncertainty of the environment. In this paper we deal with more practical aspects, in particular with the concrete applicability of the coverage stability property.

5.4.2.2 Enforcing Coverage Stability

The definition of coverage stability for an application (instead of for a single property) simply consists in the expression of coverage stability for all timeliness properties defined within the application. But not all applications can benefit from this property.

A class that can indeed benefit is what we define as the **Time-Elastic Class (Tε)** [23]. In practical terms, Tε applications are those whose bounds can be increased or decreased dynamically, such as QoS-driven applications. As already mentioned, provided that correct mappings are used to express application specific QoS requirements as timeliness requirements, it is possible to include these applications in the Tε class and possibly enforce their coverage stability.

Coverage stability is a useful property in the presence of uncertain timeliness. It means that an application has the guarantee that its timeliness properties will hold during execution with a constant probability. The trade-off is that they must be time-elastic, which means that they must allow run-time adaptation of their timing parameters.

To explain how can a Tε application achieve coverage stability under certain conditions, we must reason in terms of the services provided by the TCB and of what can be done with them. In this case, the relevant service is the duration measurement

service that may provide the necessary information to construct histograms of the distribution of durations and thus gathering evidence about coverage.

Note that a timeliness property implies that a given action has always to be performed within certain time bounds. Note also that the TCB is capable of observing those executions and measure their duration. Therefore, provided that there is a minimum number n_0 of *observed durations*, *T*, representing the *specified duration bound*, $\top$ (derived from the timeliness property), it is possible to build a discrete probability distribution function *pdf* that represents the actual distribution of the timing variable with an error p_{dev0} [13]. This error depends on the measurement error introduced by the TCB and on the error introduced by the method used to build the *pdf*. With the *pdf* one can determine the probability $P=pdf(T)$, such that being *p* the actual probability of *T*, $| p - P | \leq p_{dev0}$.

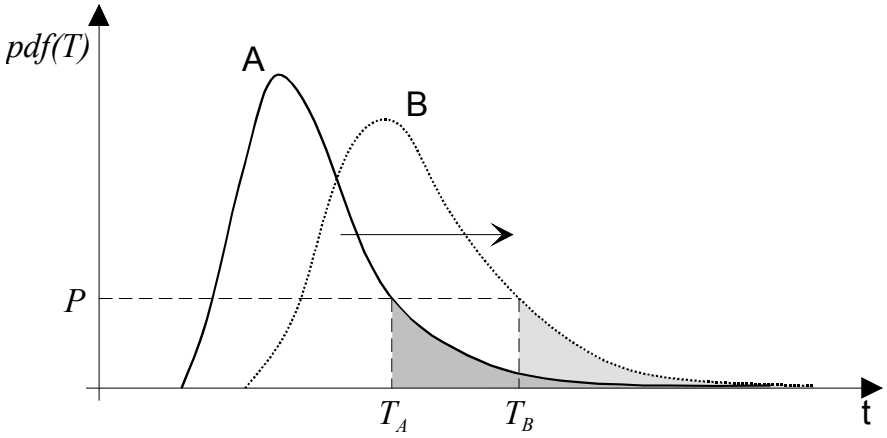


Figure 1: Example variation of distribution *pdf*(*T*) with the changing environment

In systems of uncertain timeliness, the *pdf* of a duration varies with time. Therefore, if two distinct intervals of observation are considered, what can be observed is a shift of the baseline *pdf*, as depicted in Figure 1, by curves A and B. Note that at this point we are not assuming any particular distribution, and so the curves depicted in the figure merely represent any two possible *pdf*s. Although it is enough to know that p_{dev} remains bounded, one wishes to keep the error p_{dev} small in order to predict the probability of any *T* accurately, even with periods of timeliness instability. To achieve this goal it is necessary to periodically rebuild the *pdf* that represents the actual distribution. One can take the previous *pdf*, $pdf_{i-1}(\top)$, or at least a part of that history (the most recent subset of values used to compute $pdf_{i-1}(\top)$), and the immediately subsequent *n* observed durations, and compute a new $pdf_i(\top)$ that reflects more accurately the current system state and forces p_{dev} to remain bounded and small.

It may not be practical to recompute a new *pdf* after every observed duration ($n=1$), since this may require too many computational resources without considerable benefits. However, if the value of *n* is set too high and if the environment presents large variations, the accuracy of $pdf(\top)$ can degrade and the error p_{dev} will have to be higher. The adequate value of *n*, as well as the history size, depends in fact on the behaviour of the environment, and cannot be assigned an optimal value by default. What may eventually be done is the implementation of additional mechanisms to

detect or learn particular behaviours and adjust the critical values accordingly to them (see section 6).

Having described the general principle that allows a *pdf* to be constructed with the help of a TCB service, we still have to discuss how can this *pdf* be used to enforce the coverage stability property of a Time-Elastic application.

The basic idea is simple and relies on the assumption that QoS requirements may be specified in terms of <*bound,coverage*> pairs. This means that application are constructed assuming that each time bound holds with a certain coverage. When the environment degrades, the only way in which it is possible to maintain the coverage is to assume a different bound. This is where the availability of a *pdf* becomes useful, namely for Time-Elastic applications which are able to change the bounds dynamically. It allows to directly determine the new bound that has to be used in order to maintain the coverage. In the example of Figure 1, when the environment changes and the new *pdf* B is obtained, the application maintains the degree of coverage by adapting the bound from T_A to T_B. Note that these remarks are also true when the environment gets faster: the bound should get back to its lower value as soon as possible.

A pertinent question that could be made at this point is about the entity responsible for building the *pdf*: it could be the TCB itself, or the application, or a middleware layer specifically designed for that purpose. We introduce the QoS coverage service as the logic entity in the system responsible for handling the coverage related issues. This service is presented in the next section.

5.5 The QoS Coverage Service

In a general sense, the QoS coverage service can be described as providing to applications the ability of dependably decide how to adapt time bounds in order to maintain a constant coverage level. Figure 2 illustrates the overall aspect of a system with QoS oriented services and a TCB extended with a QoS coverage service.

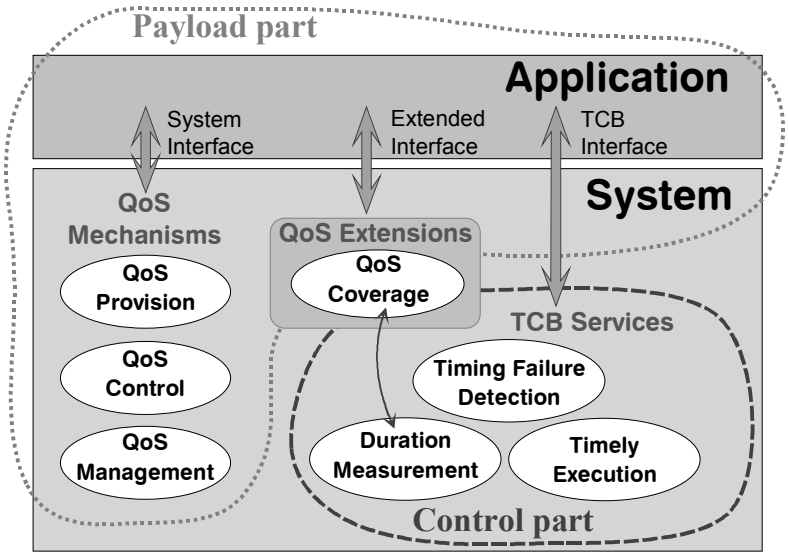


Figure 2: QoS extensions for the TCB.

Applications are layered on top of the system, which provides a set of services through dedicated interfaces. We distinguish three interfaces: the basic system interface, which provides access to all system services, including those related with QoS provision, control and management; the TCB interface that we mentioned in Section 3; and the QoS coverage service interface, which is presented below. The payload part of the system includes everything except the TCB, which constitutes the control part of the system. As we will see, the QoS coverage service can reside in any of these two parts.

5.5.1 Service Interface

We have designed an interface for the QoS coverage service that being as simple as possible yet allows the fundamental objective to be achieved. Obviously, since this objective consists in maintain the coverage of a given timeliness property, the service has to know, at least, the *bound* that must be observed and its respective desired *coverage*. In the interface functions presented in Table 2 these parameters are represented, respectively, by `bound` and `cov`. Unchanged API portions are printed in grey.

In the framework that we have defined so far, it is possible to identify two kinds of bounds. In fact, the nature of the bounds associated to actions executed locally is different from the bounds that are imposed to actions executed among distributed nodes. For instance, this difference is quite explicit in the way durations are measured by the TCB. Because of that we have explicitly defined two different functions to request a QoS coverage service: `monDistr` is used to monitor the coverage of distributed durations and `monLocal` is used for local durations.

QoS Coverage

```
id ← monDistr (bnd, cov, dev, c_id, hdlr)
id ← monLocal (bnd, cov, dev, f_id, hdlr)
new_bound ← waitChange (id)
```

Timing Failure Detection (modified functions)

```
id ← send (c_id, send_ts, spec, handler)
id,deliv_ts ← receive (c_id)
id, dur1,faulty1 Ö durn,faultyn ← waitInfo(c_id)
```

Duration Measurement (modified functions)

```
id ← startMeasurement (start_ts, f_id)
```

Table 2: Extended and modified API.

The duration of a distributed action always includes the delay for transmitting a message. Therefore, distributed durations can be observed as a result from messages sent through some communication channel using the modified `send` function presented in above table. The identifier of the channel, `c_id`, is necessary for the

coverage service to associate a given <*bound,coverage*> pair to messages transmitted through that channel. This `c_id` parameter was omitted on purpose in Table 1, but is now presented for completeness reasons. Other irrelevant parameters (for the service) are still omitted.

Similarly to distributed durations, which are associated to communication channels, local durations are associated to the execution of specific functions or parts of the code. Therefore, when requesting the QoS coverage service to monitor the coverage of a local action it is necessary to provide some identification of that action. This is done through the `f_id` identifier. Naturally, this identifier has also to be used when requesting the measurement of durations, so that these measurements can be associated to specific actions (and their respective time bounds).

For the QoS coverage service to work properly it is necessary that it knows when to inform the application that a bound needs to be adapted. It would be possible to let the service provide information whenever a new *pdf* was built, independently of the variation degree. But in the case of applications that define QoS levels and are able to adapt among these levels, this would cause unnecessary performance degradation because for small variations nothing would be done (yet an indication would be delivered to the application). To prevent this situation, both interface functions contain the `dev` parameter. It is used to indicate the deviation relative to the current bound that triggers the delivery of a QoS change indication to the application. With this parameter it is possible to configure the hysteresis of the triggering of the indications.

The last parameter, `hdlr`, can be used to provide a handler for a function that should be executed whenever a QoS change is detected. The utility of providing an handler is highly determined by the timeliness properties that are enjoyed by the QoS coverage service, but this is discussed below. On return, both monitoring functions provide an identifier, `id`, for the observed duration.

Note that independently of whether an handler is provided or not, the service always sends an indication to the application. In this interface we assume that the application is responsible for consuming and processing all indications sent by the QoS coverage service, using for that the `waitChange` function call. This function returns the most recent information concerning the duration identified by `id`, that is, it returns the bound that should be used in order to keep the desired coverage.

5.5.2 Service Operation

The service collects information relative to durations, builds the *pdf*, determines if it is necessary to inform the application of any considerable change and, finally, executes the appropriate actions.

Gathering information about distributed durations is done using the `waitInfo` function. Durations of local actions are explicitly observed using the modified duration measurement API function. The service keeps track of a certain amount of observed durations for each channel (`c_id`) and function (`f_id`). How the *pdf* is

actually built depends on the particular implementation and on some assumptions about the environment. Therefore, this is discussed in Section 6. Having some representation of the *pdf*, the current (the last reported) bound and the maximum allowed deviation, the service can easily determine if the deviation has been exceeded whenever a new *pdf* is built. If so, the new bound is recorded and reported to the application. If an handler has been specified it will be executed in the context of the QoS coverage service (whichever context this is). Whether the handler is timely executed or not depends on the timeliness properties of the service.

5.5.3 Timeliness Issues

The QoS coverage service has been so far defined as a service laying between the application and the TCB. But the consequences of implementing it in the *control* part of the system (inside the TCB) or in the *payload* part are quite different and relevant.

If the QoS coverage service is implemented as an additional TCB service, then it will obviously benefit from the fact that the TCB is a synchronous component. Every operation will be executed within known time bounds, in particular all those that are necessary for the detection of QoS changes. This means that QoS change handlers will be timely executed allowing timely reactions to QoS changes. We say that a service with these characteristics allows for **real-time, dependable adaptation**. The trade-off for having this real-time behaviour is that more complexity is being added to the TCB, increasing the probability of timeliness violations within the TCB itself, and making the TCB a "less synchronous" component than it was before. To decide whether it is worth implementing the QoS coverage service inside the TCB depends on the application and on the relative benefits that it could obtain by timely adapting to QoS changes.

If implemented in the payload part of the system, the QoS coverage service will behave according to the properties of the payload. This means that nothing can be assumed with respect to its timeliness. However, since the information that is used to build the *pdf* is still obtained using TCB services, it is possible to have certain guarantees about the (logical) correctness of the results obtained using this information. For instance, it is possible to ensure that given an interval of observation only the information relative to this interval is used to construct the correspondent *pdf*. It is also possible to ensure that all components of a distributed application have the same view of the environment. This is because the TCB delivers exactly the same information about distributed durations to all participants to which this information is relevant. The result is that applications can dependably (but possibly not timely) make decisions based on that information. The service allows for **dependable adaptation** in response to changing conditions of the environment.

We have seen that it is possible to specify QoS adaptation handlers when issuing requests to the QoS coverage service. However, the kind and complexity of the operations that can be done by these handlers depends on the location of the QoS coverage service. If the service is inside the TCB, it will only allow simple operations to be executed in order to preserve the timeliness of the TCB. There is a TCB admission control layer that verifies the feasibility of the request. When the request is accepted the handler executes in the context of the TCB, benefiting from its

synchrony properties. The practical implications of having two different execution contexts, the application and the TCB one, depend on the concrete implementation. It is possible to employ specific mechanisms to share parts of these contexts and still preserve the temporal integrity of the TCB.

5.5.4 Extending the Interface

As already stated, we followed the principle of simplicity when proposing the interface for the QoS coverage service. Now we describe an important extension that can be done in order to make the service more flexible. We first recall the reader that the QoS coverage service was devised with the main objective of keeping the coverage close to the assumed value, with obvious repercussions in the proposed interface. We also recall that we have considered a particular class of applications, the Tε class, containing applications with the ability to dynamically adapt their time bounds. However, we could as well have considered another class of applications, a coverage-elastic class (Cε), which includes applications that are able to withstand with variations of the assumed coverage values, keeping constant time bounds. These applications do not have to enjoy the coverage stability property. Instead, with the help of an extended QoS coverage service they can possibly enjoy a *Coverage Awareness* property, that is, the ability to know at a given moment the coverage value associated to some bound. Note that they still have to be constructed assuming that each time bound holds with a certain coverage, which means that QoS is still specified in terms of <*bound,coverage*> pairs. With coverage awareness these applications can employ specific adaptation procedures to handle coverage variations. For instance, they can take actions as simple as reporting the fact to the user or as complex as launching new application replicas in response to coverage degradation periods.

To accommodate this coverage-elastic class of applications, in addition to the time-elastic class, the QoS coverage service interface has to be slightly extended. The idea is to have a service that operates in one of two modes: with constant bounds or with constant coverage values. Therefore, the interface must allow for an operation `mode` to be specified and must be able to interpret the deviation parameter `dev` accordingly. On the other hand, depending on the selected mode, the `waitChange` function must be able to return either a new bound or a new coverage value.

5.6 Implementation Issues

The implementation of the QoS coverage service encompasses several issues, which can be discussed individually according to their specific functionalities. In this section we discuss some of these issues, in particular those related with the construction of the *pdf*, which we believe to be more relevant for the reader.

From a practical point of view, to implement a QoS coverage service it is necessary to decide which concrete algorithms and values will be used. For instance, since a *pdf* is built using a certain number of observed durations it is necessary to decide which number will this be.

We propose a method to build *pdf*s and detect QoS changes that relies on the following assumptions:

Probabilistic behavior – We are observing the duration of actions executed in the payload part of the system. Generically, the payload part of the system can be of any synchronism, which means that it might by synchronous, but also completely asynchronous. Consequently, there is nothing that formally limits the time it takes for actions to be executed. However, we know that when the environment is stable the execution time of specific actions usually follows some probabilistic distribution. Therefore, arbitrary behaviours can be disregarded. Nevertheless, since the execution conditions may vary, a certain probabilistic behaviour can be suddenly affected and may be transformed into a different probabilistic behaviour. This is why we assume that durations can follow any *probabilistic distribution*, and that it is not necessary to know which distribution this is.

Recognition abilities – The ideal situation would be to know the exact probability distribution associated to a certain duration. We know that this distribution varies and that it depends, among other factors, on the application behaviour and on the background execution context. Therefore, it would eventually be possible to apply additional run-time mechanisms in order to *recognize* at a given moment the probabilistic distribution more closely suited to the observed durations. However, we assume that we do not have enough computational power to do that during the execution.

Regular execution – We have mentioned that coverage should remain stable over an interval of mission. The idea is to provide a service that is well dimensioned for those intervals. We assume that applications have a regular execution, which allows the clear identification of intervals of mission. We further assume that an interval of mission contains a sufficient number of observable actions, required to construct a *pdf*. Coverage assumptions cannot be guaranteed to hold for sporadic actions.

Since we assume that the probabilistic distribution of durations in unknown, we will show that it is possible to determine <*bound,coverage*> pairs for a duration D, using only the expected value $E(D)$ and the variance $V(D)$ relative to that duration. Then we will describe how to compute $E(D)$ and $V(D)$.

We use a known result of probability theory, the *One-sided Inequality* (for instance, *see* [2]), which states that for any random variable D with a finite expected value $E(D)$ and a finite variance $V(D)$ we can bound the probability that D is higher than some value t:

$$P(D > t) \leq \frac{V(D)}{V(D) + (t - E(D))^2} \text{ , for all } t > E(D)$$

This expression can be used to calculate an upper bound for the probability of a time bound t being violated. This also corresponds to a lower bound for the coverage of the assumed bound t. For an assumed minimum coverage $C_{\min}$, one can find the time bound t that has to be used in order to guarantee $C_{\min}$:

$$t = \frac{2E(D) + \sqrt{4E(D)^2 - 4(E(D)^2 + V(D) - \frac{V(D)}{1 - C_{\min}})}}{2}$$

Obviously, if the objective is to keep constant bounds and simply be aware of the maximum possible coverage for a given bound (*see* Section 5.4), the expression to use has to be:

$$C_{\min} = 1 - \frac{V(D)}{V(D) + (t - E(D)))^2}$$

Estimating *E*(*D*) and *V*(*D*)

The values of $E(D)$ and $V(D)$ are *estimated* using a finite number of observed durations. The idea is simply to have a set of measured durations and then obtain the **average** and the **variance** corresponding to that set. The size of the set should be determined by the typical interval of mission of the application which is using the service. Note that intuitively it may seem a better approach to use as many values as possible when estimating $E(D)$ and $V(D)$, even if they are outside the interval of mission. However, since we assume that the environment (and consequently the distribution) may not remain stable, by using values observed outside the interval of mission we may just be degrading the accuracy of the estimated distribution.

Method accuracy

The accuracy of the proposed method is influenced by three kinds of errors. In the first place, the values used to estimate $E(D)$ and $V(D)$ (provided by the duration measurement service) have an associated error. Although this error is not explicitly provided in the interface (the measured durations are upper bounds), it would be possible to have a duration measurement service returning <*measurement,error*> pairs instead of simply <*measurement_upper_bound*>.

In the second place, there is the error associated with the estimation of $E(D)$ and $V(D)$. This error obviously depends on the number of values used to obtain the estimation, which, as we have seen, should not be arbitrarily high by including values observed outside a certain interval of mission.

Finally, when using the proposed expressions derived from the one-sided inequality, we are introducing an error that depends on the "real" distribution corresponding to the duration. Since we do not assume any particular distribution, the formulas provide pessimistic but *secure* bounds, that in any case can compromise the system safety. Note that by removing the second assumption presented above (recognition abilities), it would become possible to consider specific distributions for the probabilistic behaviour of durations (e.g. exponential or normal) and avoid this last source of errors.

5.7 Conclusions

We have presented a new approach for QoS adaptation, developed in the context of the Timely Computing Base (TCB) model. We addressed the problem of providing an adequate framework and programming infrastructure to applications with QoS requirements (specifically expressed in terms of timeliness properties), running in unpredictable environments.

We introduced an innovative analysis of the effect of timing failures on application correctness. Besides the obvious effect of delay, we identified a long-term effect, of decreased coverage of assumptions, and an instantaneous effect, of contamination of other properties. Even when delays are allowed, any of these effects can lead to undesirable behaviour of a system. We addressed from a practical perspective the effect of decreased coverage, and showed that there is a class of applications, the time-elastic class, which, with the help of the TCB, can dependably adapt and enjoy the property of coverage stability. We presented the QoS coverage service and discussed its synchrony properties. In particular, we compared the properties of a TCB based and a payload based QoS coverage service. We also discussed possible modifications to the interface in order to address other classes of applications as, for instance, the coverage-elastic class. Finally, a few important issues relative to the concrete operation of the QoS coverage service have also been discussed.

As future work we intend to extend our RT-Linux TCB [9] with the ideas presented in this paper. The results of this implementation will be reported in another paper.

5.8 References

[1] T. F. Abdelzaher and K. G. Shin. End-host architecture for QoS-adaptive communication. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, Denver, Colorado, USA, June 1998.

[2] A. O. Allen. *Probability, Statistics, and Queueing theory with Computer Science Applications*. Academic Press, 2nd edition, 1990.

[3] C. Almeida and P. Veríssimo. Using light-weight groups to handle timing failures in *quasi-synchronous* systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*. Pages 430-439, Madris, Spain, Dec. 1998.

[4] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems Journal - Special Issue on QoS Architecture*, 6(3):138-151, May 1998.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An architecture for differentiated services*, December 1998.

[6] R. Braden, D. Clark, and S. Shenker. *Integrated services in the internet architecture: an overview*, June 1994.

[7] I. Busse, B. Deffner, and H. Schulzrinne. Dynamic QoS Control of Multimedia Applications based on RTP. *Computer Communications*, 19(1), January 1996.

[8] A. Campbell and G. Coulson. A QoS adaptive transport system: Design, implementation and experience. In *Proceedings of the Fourth ACM Multimedia Conference*, pages 117-128, New York, USA, November 1996.

[9] A. Casimiro, P. Martins, and P. Veríssimo. How to build a Timely Computing Base using Real-Time Linux. In *Proceedings of the 2000 IEEE Intl. Workshop on Factory Communication Systems*, pages 127-134, Porto, Portugal, September 2000.

[10] S. Chatterjee, J. Sydir, B. Sabata, and T. Lawrence. Modeling applications for adaptive qosbased resource management. In *Proceedings of the 2nd IEEE High Assurance Engineering Workshop*, Bethesda, Maryland, USA, August 1997.

[11] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, pages 642-657, June 1999.

[12] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, D. Bakken, M. Berman, D. Karr, and R. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, West Lafayette, Indiana, USA, October 1998.

[13] W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, New York, 2nd edition, 1971.

[14] P. Ferguson and G. Huston. Quality of Service: *Delivering QoS on the internet and in corporate networks*. John Wiley & Sons Inc., 1998.

[15] C. Fetzer and F. Cristian. Using fail-awareness to design adaptive real-time applications. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, Dayton, Ohio, USA, July 1997.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, April 1985.

[17] I. Foster, V. Sander, and A. Roy. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the Eighth International Workshop on Quality of Service*, pages 181-188, Westin William Penn, Pittsburgh, USA, June 2000.

[18] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, 17(9):1632-1650, September 1999.

[19] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer. Issues in managing soft QoS requirements in distributed systems using a policy-based framework. In *Proceedings of the International Workshop, POLICY 2001*, LNCS 1995, pages 185-201, Bristol, UK, January 2001.

[20] D. Powell. Failure mode assumptions and assumption coverage. In *Digest of Papers, The 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 386-395, Boston, USA, July 1992.

[21] F. Siqueira and V. Cahill. Quartz: A QoS architecture for open systems. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS 2000),* pages 197-204, Taipei, Taiwan, April 2000.

[22] P. Veríssimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the TCOS*, 7(4):35-39, Winter 1995.

[23] P. Veríssimo and A. Casimiro. The Timely Computing Base. DI/FCUL TR 99-2, Department of Computer Science, University of Lisboa, April 1999. Short version appeared in the *Digest of Fast Abstracts, The 29th IEEE Intl. Symposium on Fault-Tolerant Computing*, Madison, USA, June 1999.

[24] P. Veríssimo, A. Casimiro, and C. Fetzer. The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533-542, New York, USA, June 2000. IEEE Computer Society Press.

[25] C. Vogt, L. C. Wolf, R. G. Herrtwich, and H. Wittig. Heirat - qualityof- service management for distributed multimedia systems. *Special Issue on QoS Systems of ACM Multimedia Systems Journal*, 6(3):152-166, May 1998.

[26] X. Wang and H. Schulzrinne. Comparison of adaptive internet multimedia applications. *IEICE Transactions*, E82-B(6):806-818, June 1999.

[27] D. Xu, D. Wichadakul, and K. Nahrstedt. Multimedia service con_guration and reservation in heterogeneous environments. In *Proceedings of International Conference on Distributed Computing Systems*, Taipei, Taiwan, April 2000.